High-Availability Cluster Support for IBM Informix Dynamic Server (IDS) on Linux

by

Lars Daniel Forseth

A thesis submitted in partial fulfillment
of the requirements for the degree of

Diplom-Informatiker (Berufsakademie)

in the Graduate Academic Unit of

Applied Computer Science

at the

Berufsakademie Stuttgart

September 2007

Duration:	3 months
Course:	TITAIA2004
Company:	IBM Deutschland GmbH
Examiner at company:	Martin Fuerderer
Examiner at academy:	Rudolf Mehl

High-Availability Cluster Support for IBM Informix Dynamic Server (IDS) on Linux

by

Lars Daniel Forseth

A thesis submitted in partial fulfillment
of the requirements for the degree of

Diplom-Informatiker (Berufsakademie)

in the Graduate Academic Unit of

Applied Computer Science

at the

Berufsakademie Stuttgart

September 2007

Duration:	3 months
Course:	TITAIA2004
Company:	IBM Deutschland GmbH
Examiner at company:	Martin Fuerderer
Examiner at academy:	Rudolf Mehl

Selbständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema

„High-Availability Cluster Support for IBM Informix Dynamic Server (IDS) on Linux“

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Stuttgart, 27.08.2007

_____

(Lars D. Forseth)

English Version of the above statement:

I hereby certify that this diploma thesis with the theme “High-Availability Cluster Support for IBM Informix Dynamic Server (IDS) on Linux” does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

Abstract

The availability of database servers is fundamental for businesses nowadays. A downtime of database server for a day can cost a company thousands of dollars or even more. Therefore so called High-Availability (HA) cluster systems are set up to guarantee a certain amount of availability by redundancy. IBM Informix Dynamic Server (IDS) is one of the two leading database management systems (DBMS) IBM offers. There exists a proprietary HA cluster solution for Sun Solaris and an HA solution via replication on application level. In order to extend the HA portfolio of IDS, an Open Source or at least as cheap as possible HA cluster solution on Linux is desired. After a theoretical overview on clustering and HA clusters in general, this thesis analyzes different HA cluster software products for Linux, chooses one and describes the implementation and validation of developing a resource agent for IDS for the Open Source HA clustering software project Linux-HA aka Heartbeat. As an additional result, installation tutorials on how to set up the virtual three-node test cluster on Suse Linux Enterprise Server 10 (SLES10) and Red Hat Enterprise Linux 5 (RHEL5) that is used for the validation process are written and appended.

This thesis assumes that the reader has an understanding of the Linux operating system, networking in general, and good knowledge of shell scripting and basic experiences with database servers.

Table of Contents

Contact Information iv

Acknowledgements v

Trademarks and Product License vi

List of Figures x

List of Listings xi

List of Tables xii

Table of Abbreviations xiii

Introduction xvii

PART I – THEORETICAL ANALYSIS 1

1. Clusters in General 2

1.1. Cluster Term Definition 2

1.2. Cluster Categories / Types 6

1.3. High-Availability (HA) 8

2. IBM Informix Dynamic Server (IDS) 13

2.1. General Overview 13

2.2. HA Solutions for IDS 17

3. HA Cluster Software Products for Linux 23

3.1. Overview on HA Cluster Software for Linux 23

3.2. Choosing an HA Clustering Product 34

4. Distributed Replicated Block Device (DRBD) 36

5. Linux-HA 39
5.1. Heartbeat Version 1 Configuration Mode 40
5.2. Heartbeat Version 2 – Features and Configuration 43
5.3. Heartbeat Version 2 – STONITH, Quorum and Ping Nodes 47
5.4. Heartbeat Version 2 – Components and their Functioning 49
5.5. Resource Agents and their Implementation 55

PART II – DEVELOPMENT AND VALIDATION PROCESS 60

6. Implementing the IDS Resource Agent for Linux-HA 61
6.1. Initial Thoughts and Specifications 61
6.2. Development Environment 63
6.3. Structuring of the IDS RA in Detail 66
6.4. Issues and Decisions during Development 71
6.5. First Tests during the Development Process 72

7. Validating the IDS Resource Agent for Linux-HA 74
7.1. Purpose of the Validation Process 74
7.2. Validation Environment 75
7.3. Tests run during the Validation Process 80
7.4. The IDS Transaction Validation Script (ITVS) 82
7.5. Validation Test Results 86
7.6. Issues and Decisions during Validation 86

PART III – RESULTS AND OUTLOOK 90

8. Project Results 91

9. Project Outlook 92

PART IV – APPENDIX 93

A. Project Specifications 94

A.1. Non-Functional Requirements Specification (NFRS) 94

A.2. Functional Requirements Specification (FRS) 95

A.3. Design Specification (DS) 107

A.4. Test Cases (TCs) 118

B. GNU General Public License, Version 2 131

C. Bibliography 136

D. CD-ROM 144

Contact Information

The following table presents the most important persons involved with this thesis:

Student and Author of the thesis	Lars Daniel Forseth Student of Applied Computer Science at IBM Germany lars@forseth.de +49 – 176 – 20 10 31 01
Tutor of the thesis **within IBM Germany**	Martin Fuerderer IBM Informix Development Munich, Germany Information Management martinfu@de.ibm.com +49 – 89 – 4504 – 1421
Tutor at the University of Applied **Sciences Stuttgart, Germany**	Rudolf Mehl Alcatel-Lucent Deutschland AG rudolf.mehl@alcatel-lucent.de

Acknowledgements

In the first place, I would like to thank IBM Germany for giving me a three year contract as a student of applied computer science at the University of Cooperative Education in Stuttgart, Germany (http://www.ba-stuttgart.de). The three years of switching between theoretical phases at the university and project work within IBM Germany did not only improve my technical skills, but also highly emphasized my personal development for becoming a computer scientist. Details (in German though) on IBM's cooperative studies programs can be found at the following URL: http://www.ibm.com/employment/de/schueler/.

The persons to thank, right after the company itself, are of course my tutors at IBM for this thesis and their team "Informix Development, Munich". The team members are in particular: Martin Fuerderer, Sandor Szabo, Karl Ostner and Andreas Breitfeld. They have supported me in all technical and moral means I could think of. So thank you guys, for everything!

I would also like to thank Alan Robertson and the Linux-HA community on the IRC channel and the mailing lists for supporting when I ran into troubles with Heartbeat. This is also a good time to thank Alan once again for publishing the IDS resource agent as part of the Heartbeat package for me and for sending me a HA-t (see http://www.linux-ha.org/HAt) via snail mail.

A thank you also goes to the author of "In Search of Clusters", Greg F. Pfister, who was so kind to provide me dog cliparts for the figure comparing a single dog, a pack of dogs and a "Savage Multiheaded Pooch" in the chapter on clusters in general.

The users in the IRC channels #rehl, #suse and #vbox on irc.freenode.net were also a great help in resolving several bugs and difficulties while setting up the virtual validation cluster. So, a thank you also goes to them.

Last but not least, I would like to thank my family and friends for giving me moral assistance and strength! The persons I think of will know they are meant here…

Trademarks and Product License

Trademarks

IBM and the IBM logo, IBM Informix Dynamic Server (IDS), AIX, DB2, IBM Informix Data Blades, Redbooks, PowerPC, xSeries and High Availability Cluster Multiprocessing (HACMP) are registered trademarks or trademarks of the International Business Machines Corporation in the United States and other countries.

Lenovo and ThinkPad are registered trademarks or trademarks of Lenovo in the United States and other countries.

Solaris, Sun Cluster, Java and JDBC are registered trademarks or trademarks of Sun Microsystems, Inc. in the United States and other countries.

SPARC is a registered trademark of SPARC International, Inc. in the United States and other countries.

Red Hat, Fedora, Red Hat Enterprise Linux (RHEL) and Red Hat Cluster Suite (RHCS) are registered trademarks or trademarks of Red Hat, Inc. in the United States and other countries.

Suse and Suse Linux Enterprise Server (SLES) are registered trademarks or trademarks of Novell, Inc. in the United States and other countries.

Hewlett-Packard, HP-UX and HP Serviceguard are registered trademarks or trademarks of Hewlett-Packard Company in the United States and other countries

Oracle is a registered trademark of Oracle Corporation in the United States and other countries.

SAP is a trademarks or registered trademarks of SAP AG in Germany and other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Linux is a trademark of Linus Torvalds in the United States and other countries.

Intel, Pentium and Xeon are registered trademarks or trademarks of Intel Corporation in the United States and other countries.

3Com is a registered trademark or trademark of 3Com Corporation in the United States and other countries.

Windows and Windows Server either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

Distributed Replicated Block Device (DRBD) is a registered trademark or trademark of LINBIT Information Technologies GmbH in Austria and other countries.

Slackware is a registered trademark of Patrick Volkerding and Slackware Linux, Inc. in the United States and other countries.

Mandrake is a registered trademark or trademark of Mandriva in the United States and other countries.

Google is a registered trademark or trademark of Google, Inc. in the United States and other countries.

Debian is a registered trademark of Software in the Public Interest, Inc. in the United States and other countries.

Veritas and Veritas Cluster Server are registered trademarks or trademarks of the Symantec Corporation in the United States and other countries.

Apache and Apache Webserver are registered trademarks or trademarks of the Apache Software Foundation in the United States and other countries.

Apple and Mac OS X are registered trademarks or trademarks of Apple, Inc. in the United States and other countries.

MySQL is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Red Flag is a registered trademark or trademark of Red Flag Software Co., Ltd. in China and other countries.

Ubuntu is a registered trademark of Canonical Ltd. in the European Union and other countries.

Gentoo is a registered trademark or trademark of Gentoo Foundation, Inc. in the United States and other countries.

FreeBSD is a registered trademark of the FreeBSD Foundation in the United States and other countries.

SGI is a trademark of Silicon Graphics, Inc. in the United States and other countries.

Broadcom and NetXtreme are registered trademarks or trademarks of Broadcom, Corp. in the United States and other countries.

VirtualBox and innotek are registered trademarks or trademarks of innotek GmbH in Germany and other countries.

Other company, product, and service names used in this publication may be registered trademarks, trademarks or service marks of others. They are respectfully acknowledged here if not already included above.

Product License

The IDS OCF resource agent and wrapper script are licensed under the GNU General Public License, Version 2 and later. A copy of the license is attached in Appendix B.

List of Figures

Figure 1: Single Dog, Pack of Dogs, Savage Multiheaded Pooch and a Rabbit.....3
Figure 2: Three-Tiered Distributed System.....4
Figure 3: Cluster as a Part of a Distributed System.....5
Figure 4: IDS Fragmentation 16
Figure 5: How HDR works 18
Figure 6: Components Diagram of Sun Cluster 3.x 21
Figure 7: DRBD Two-Node Cluster 37
Figure 8: Two-Node Cluster with STONITH Device 48
Figure 9: Heartbeat Version 2 – Process Tree View 50
Figure 10: Heartbeat Version 2 – Data Flow 54
Figure 11: Development Environment Graph 65
Figure 12: Validation Environment without considering Virtualization 77
Figure 13: Validation Environment considering Virtualization.....79
Figure 14: ITVS Transaction Timeline 84

List of Listings

Listing 1: Sample ha.cf Configuration File – Version 1 Configuration Mode 40
Listing 2: Sample haresources Configuration File 40
Listing 3: Sample authkeys Configuration File.....40
Listing 4: Sample ha.cf Configuration File – Version 2 Configuration Mode 43
Listing 5: Sample Initial cib.xml Configuration File 45
Listing 6: Sample resources Sub-Section of the CIB.....45
Listing 7: Sample constraints Sub-Section of the CIB 46
Listing 8: Basic Sample OCF Resource Agent.....57
Listing 9: Extended Sample OCF Resource Agent.....58
Listing 10: Usage Description of the ITVS.....82
Listing 11: SQL Statements of the Transaction t4 84
Listing 12: ITVS Output when successfully passing the Parameter “test-before” 85
Listing 13: ITVS Output when successfully passing the Parameter “test-after” 85

List of Tables

Table 1: Availability Representation in Percent 10
Table 2: Comparison Table of HA Cluster Software Products for Linux 35
Table 3: Hardware Specifications of the Development Environment 64
Table 4: Server Hardware Specifications for the Validation Environment 75
Table 5: IBM ThinkPad Hardware Specifications for the Validation Environment 79
Table 6: Validation Test Results Summarization Table 86

Table of Abbreviations

Abbreviation	Meaning
aka	also known as
API	Application Programming Interface
BLOB	Binary Large Object
BSD	Berkeley Software Distribution
CCM	Consensus Cluster Membership
CD-ROM	Compact Disk – Read-Only Memory
CIB	Cluster Information Base
COBOL	Common Business-Oriented Language
Corp.	Corporation
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CRM	Cluster Resource Manager
CVS	Concurrent Versioning System
DBA	Database Administrator
DBMS	Database Management System
DC	Designated Coordinator
dev	Development or Developer
DHCP	Dynamic Host Configuration Protocol
DNA	Deoxyribonucleic Acid
DNS	Domain Name System
DRBD	Distributed Replicated Block Device
DS	Design Specification
DTD	Document Type Definition
DVD	Digital Versatile Disc

ER	Enterprise Replication
FAQ	Frequently Asked Questions
FRS	Functional Requirements Specification
GB	Gigabytes
GHz	Gigahertz
GNU	GNU is Not Unix
GPL	General Public License
GPS	Global Positioning System
GRUB	GNU Grand Unified Bootloader
GUI	Graphical User Interface
HA	High-Availability
HACMP	IBM High Availability Cluster Multiprocessing
HDR	High Availability Data Replication
HPC	High-Performance Computing
Hz	Hertz
I/O	Input/Output
IBM	International Business Machines Corp.
ID	Identification
IDS	IBM Informix Dynamic Server
IIUG	International Informix User Group
Inc.	Incorporation
IP	Internet Protocol
IRC	Internet Relay Chat
IT	Information Technology
ITVS	IDS Transaction Validation Script
JDBC	Java Database Connectivity
LB	Load-Balancing
LGPL	GNU Lesser General Public License
LILO	Linux Loader

LRM	Local Resource Manager
LSB	Linux Standard Base
LVM	Logical Volume Manager
MAC	Media Access Control
MB	Megabytes
Mbit/s	Megabits per second
MD5	Message Digest Algorithm 5
MHz	Megahertz
MTBF	Mean Time Between Failures
MTTR	Maximum Time To Repair
NFRS	Non-Functional Requirements Specification
NFS	Network File System
NTP	Network Time Protocol
OCF	Open Cluster Framework
OLTP	Online Transaction Processing
p.	Page(s)
PE	Policy Engine
PHP	PHP Hypertext Preprocessor
RA	Resource Agent
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RFC	Request For Comments
RHCS	Red Hat Cluster Suite
RHEL5	Red Hat Enterprise Linux 5
RPM	Red Hat Package Manager
SCSI	Small Computer System Interface
SHA1	Secure Hash Algorithm 1
SLES10	Suse Linux Enterprise 10
SNIA	Storage Networking Industry Association

SQL	Structured Query Language
SSH	Secure Shell
SSI	Single System Image
STONITH	Shoot The Other Node In The Head
TC	Test Case
TCO	Total Costs of Ownership
TE	Transition Engine
U.S.	United States (of America)
UDP	User Datagram Protocol
UDR	User Defined Routines
UDT	User Defined Types
URL	Uniform Resource Locator
USA	United States of America
VDI	Virtual Disk Image
VM	Virtual Machine
VP	Virtual Processor
VRDP	VirtualBox Remote Desktop Protocol
XML	Extensible Markup Language
Y2K	Year 2000
YUM	Yellow dog Update, Modified

Introduction

Motivation

Nowadays businesses depend more than ever on their IT infrastructure and every day's business would be more or less unachievable without laptops, desktop computers, and servers, computer networks, enterprise intranets and the Internet. Hence, the availability of these systems is very critical in today's business world. The persons responsible for managing a company's database servers, web servers, file servers, backup systems and the servers hosting enterprise applications and such are forced to guarantee a certain availability of these systems.

Setting up a high-availability cluster, also known as failover cluster, enables administrators to guarantee a certain amount of availability. In the simplest case the cluster consists of two systems and if one of the two fails, the remaining system takes over the responsibilities of the failed system. The additional goal hereby is that users do not notice this failover at all they are disconnected from the system but can immediately reconnect.

IBM Informix Dynamic Server (IDS) is one of the two primary database systems offered by IBM. IDS runs on several platforms including Linux, Microsoft Windows Server, and the major Unix derivates such as Sun Solaris. There exists a working solution to set up a high-availability cluster running IDS depending on Sun Cluster 3.x on Sun Solaris. This working solution can be used as a prototype for further analysis.

The goal of this diploma thesis is to research the different possibilities to implement a high-availability cluster on Linux and analyze them. Result of the thesis should be to choose the most convincing solution and implement a failover cluster solution for IDS. The solution should run on the two popular Linux distributions Red Hat Enterprise Linux 5 and Suse Linux Enterprise Server 10.

Structuring

Part I – Theoretical Analysis

Part I of the thesis analyses the topics needed as a theoretical preparation for the development process in Part II. Though not all chapters of Part I have to be read in order to follow the descriptions in the development process, they serve a better general understanding of the thesis' theme.

- *Chapter 1 – Clusters in General:*
  In order to understand the main topic of the thesis and what this is all about, clusters in general and the term of High-Availability (HA) are introduced.

- *Chapter 2 – IBM Informix Dynamic Server (IDS):*
  The target database server product that should be managed as a cluster resource is the IBM Informix Dynamic Server (IDS). Therefore, IDS is presented. In addition, the two HA solutions which already exist for IDS are introduced: High-Availability Data Replication (HDR) and IDSagent for Sun Cluster 3.x on Sun Solaris.

- *Chapter 3 – HA Cluster Software Products for Linux:*
  The third chapter represents the heart of the analysis process of this thesis and justifies the decision of which cluster software to pick for final implementation.

- *Chapter 4 – Distributed Replicated Block Device (DRBD):*
  As the cluster of the development environment is based on data replication instead of shared storage, the used data replication product (DRBD) is introduced here.

- *Chapter 5 – Linux-HA:*
  In the final chapter of the theoretical analysis, the HA cluster software product for Linux chosen for the development process is discussed in detail.

Part II – Development and Validation Process

Part II describes in detail how the IDS resource agent for Heartbeat was implemented and finally validated. This is the heart of the thesis.

- *Chapter 6 – Implementing the IDS Resource Agent for Linux-HA:*
  This chapter describes in detail how the IDS resource agent is implemented. In addition, the development environment and occurred issues are presented.

- *Chapter 7 – Validating the IDS Resource Agent for Linux-HA:*
  The seventh chapter presents the validation environment and the tests run for validation of the IDS resource agent and the test results. Furthermore, issues that showed up during the validation process are discussed.

Part III – Results and Outlook

Part III summarizes the results of the thesis and briefly lists once again the problems occurred during the development and validation process. In addition, a possible outlook on the project is given.

- *Chapter 8 – Project Results:*
  This chapter summarizes the final results of the thesis and project.

- *Chapter 9 – Project Outlook:*
  The ninth chapter presents a possible outlook on the project and suggests further steps which were not covered by this thesis.

Part IV – Appendix

In Part IV (the Appendix), the source code of the final IDS resource agent and all documents which are not directly a part of the thesis itself are attached.

- *Appendix A – Project Specifications:*
  The project specifications were created as a preparation for implementing and validating the IDS resource agent. They include the non-functional requirements specification (NFRS), the functional requirements specification (FRS), the design specification (DS) and the test cases (TCs) for the validation process.

- *Appendix B – GNU General Public License, Version 2:*
  The resulting IDS resource agent is published under the GNU General Public License, Version 2. Therefore a copy of this license is attached here.

- *Appendix C – Bibliography:*
  The bibliography lists all resources, both books and online resources, used for this thesis.

- *Appendix D – CD-ROM:*
  The attached CD-ROM contains all documents, configuration files, figures, source codes and other files that were created and used during the project.

Part I – Theoretical Analysis

"In a few minutes a computer can make a mistake so great that it would have taken many men many months to equal it." – Author unknown [Q1]

"The survivability of a system is directly proportional to the price of the cooling system applied to it and inversely proportional to the amount of use it sees." – Murphy’s Computer Laws [Q2]

"Complete computer breakdown will happen shortly after the maintenance person has left." – Murphy’s Computer Laws [Q2]

"Anything that can be automatically done for you can be automatically done to you." - Wyland's Law of Automation [Q3]

1. Clusters in General

1.1. Cluster Term Definition

The term cluster is often used in several meanings and there exist different definitions for it. In chemistry, a cluster is “A number of metal centers grouped close together which can have direct metal bonding interactions or interactions through a bridging ligand, but are not necessarily held together by these interactions.” [ChC01]. Furthermore in physics, “Open clusters are physically related groups of stars held together by mutual gravitational attraction.” [FroKro01]. There are many more uses of the word cluster in various sciences.

In computer science this is even worse. There exists a vast number of definitions and especially opinions of how to define a cluster. The book “In Search of Clusters” by Gregory F. Pfister [Pfi01] gives a detailed overview on the field of parallel systems, distributed systems and clusters and tries to distinguish them and give a definition of clusters. Nevertheless, this book also has difficulties to clearly make a unique definition for each of the mentioned system types as they overlap each other in several points. So uniquely defining the term cluster is and remains difficult. Despite the discussion and confusion of finding a unique definition, this chapter describes how the term cluster is used throughout this thesis.

According to Pfister, “[a] cluster is a type of parallel or distributed system that

- Consists of a collection of interconnected whole computers,
- And is used as a single, unified computing resource.” [Pfi01, p. 72]

Pfister uses the term “whole computers” here in order to refer a node to a complete computer system that has its own CPU, memory, hard disk and so on. This makes it a little bit easier to distinguish cluster from a parallel system, because in a parallel system in most cases at least one piece of hardware (i.e. the memory) is shared between its members. Symmetric multiprocessors are a famous example here: while having several CPUs, they all share the same memory, I/O buses and devices. This

does not mean parallel systems cannot consist of whole computers, though. In fact, they can and sometimes do. In this case the system is rather referred to as a cluster used for parallel programming instead of calling it a parallel system. In order to visualize the difference between a cluster and a parallel system, Pfister uses a quite amusing example comparing a single dog, a pack of dogs and a "Savage Multiheaded Pooch" (a single dog with several heads, similar to the famous monstrous female character Medusa having in the Greek mythology) [Pfi01, p. 73]. Figure 1 shows the mentioned dogs and is based on (unofficial) figures provided directly by the author of "In Search of Clusters", Greg F. Pfister. The rabbit in Figure 1 is taken from the Open Clip Art Library [OCAL01].

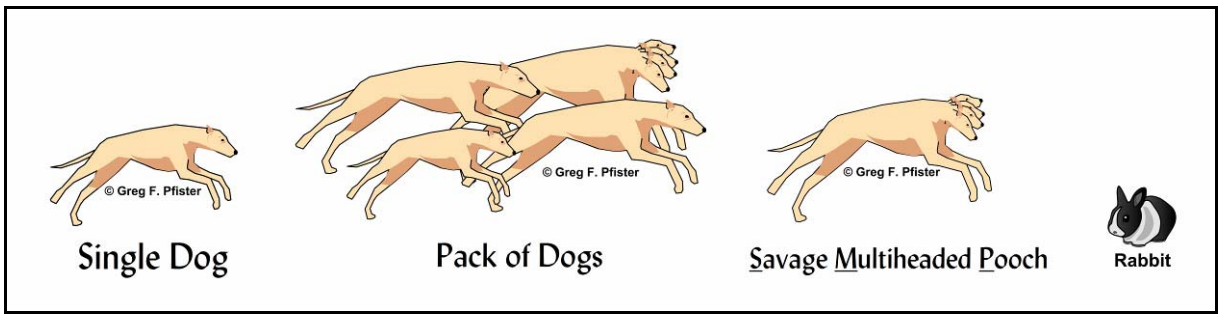


Figure 1: Single Dog, Pack of Dogs, Savage Multiheaded Pooch and a Rabbit

The abilities of a single dog are quite clear and refer to a single computer system. A pack of dogs refers here to a cluster (of dogs). Obviously a cluster of dogs can chase more rabbits at the same time or chase one single rabbit more efficiently than a single dog. Furthermore, if one dog is ill the rest of the dog cluster is not affected and can continue chasing. On the other hand, a cluster of dogs needs more food and more care than a single dog or a Savage Multiheaded Pooch. The same holds for clusters of computers and the jobs being put on them. In contrary, though a Savage Multiheaded Pooch can chase more rabbits at the same time or chase a rabbit more efficiently and needs less food and care than a cluster of dogs (to a certain degree) it certainly is affected if the pooch breaks a leg. In that case the pooch can eat food, but it definitely cannot chase rabbits anymore.

The distinction between distributed systems and clusters is more difficult than the one to parallel systems. The reason is that parts of or even the complete distributed system also can be regarded as a cluster and vice-versa. In general a distributed system consists of several so called tiers. A tier hereby is a sort of layer of the distributed system with a specific responsibility. A famous example for a two-tiered distributed system is the client-server architecture whereby the server offers a service (i.e. a web server holding web contents) to the client. In the field of web sites, the client's web browser software represents the first tier and the web server software running on the server represents the second tier. If the web server offers dynamic contents coming from a separate database server, then the database server is the third tier. Figure 2 presents this example of a three-tiered distributed system.

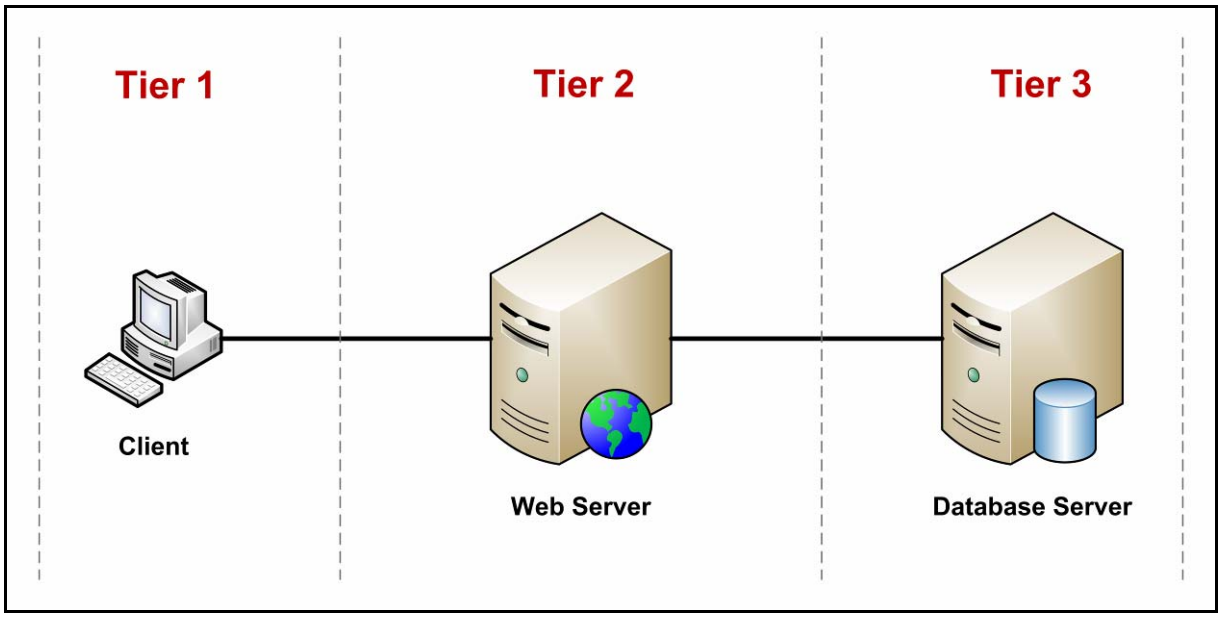


Figure 2: Three-Tiered Distributed System

Keeping this in mind, a cluster offering services to users can actually always be seen as a node in a distributed system. It is still a cluster, but it is also a part of a (larger) distributed system. Figure 3 illustrates this.

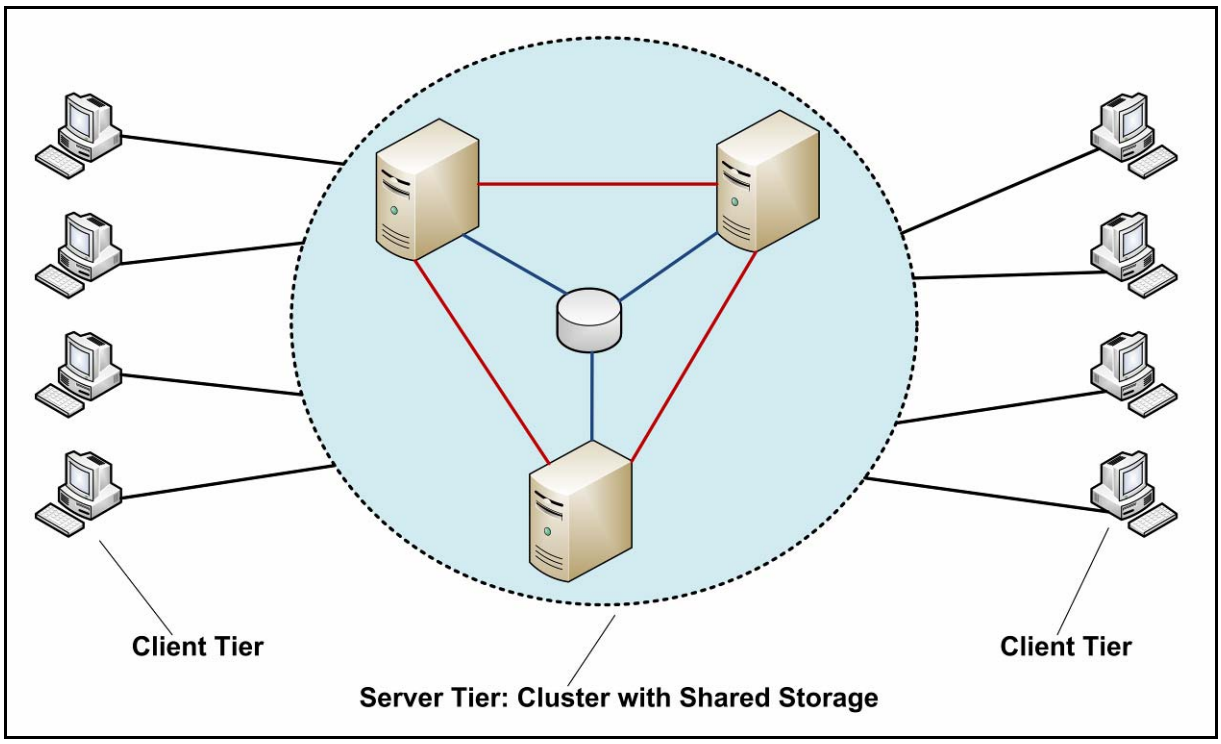


Figure 3: Cluster as a Part of a Distributed System

As mentioned before, in a distributed system each layer has a specific responsibility. This is a good indicator in order to distinguish them from a cluster: internal anonymity. The term internal anonymity means here that members of a cluster, contrary to a distributed system, usually do not have a specific responsibility assigned to them, like being just a database server or just a web server. Cluster nodes rather are treated equal among each other and do not depend on each other. Any node in the cluster could be easily replaced by another node. This anonymity has its prize, though. While in a distributed system a layer has a fixed set of IP addresses and domain names assigned to it, the nodes of a cluster are usually not directly addressable from outside the cluster. This causes an increased complexity in management. Nevertheless, this can be overcome by assigning several virtual IP addresses as resources in the cluster and if one node fails the virtual IP address of the failed node is assigned to a different node. This node is then available via two virtual IP addresses and the user does not note the difference.

For reasons of simplicity and in order to avoid a discussion filling several pages, this thesis uses the cluster for a set of separate computer machines connected together and appearing to the user as one single system. When using the term cluster in this thesis, potential similarities with a distributed system or even a parallel system are not intentional and therefore not discussed any further.

1.2. Cluster Categories / Types

There are three major usage areas for clusters:

- **High-Performance Computing (HPC) clusters**
- **Load-Balancing (LB) clusters**
- **High-Availability (HA) clusters**

High-Performance Computing (HPC) clusters are very popular and probably one of the first images that come to one's mind when hearing the term cluster: a large set of computer systems in a dark cellar having immense calculation jobs scaled among them. In fact, this is probably the oldest form of clusters and developed from scientific computation centers. A new approach to HPC clusters is the field of Grid computing where the cluster nodes are usually single (home) computer systems spread over large distances. In most cases, communication to other cluster members is not needed during each computation task. That is why grid clusters are used for projects like Folding@Home [FatH01] and World Community Grid [WCG01]. The Folding@Home project spreads DNA calculation jobs over thousands of home computers running a small background client program. The aim of Folding@Home is to find cures for diseases such as Alzheimer, cancer or Parkinson. World Community Grid has similar goals and supports several humanity projects, probably one of the reasons why it is powered by IBM.

Load-Balancing (LB) clusters consist of a hidden back-end and a front-end which accepts user requests and which is visible to the users of the cluster system. A LB cluster distributes incoming user requests by previously defined criteria among the back-end cluster nodes in order to increase the overall throughput and performance of the cluster system. The most obvious criterion thereby is the individual workload of a node: an incoming user request is forwarded to the node which currently has the lowest workload within the cluster. Other criteria may include date and time, geographical information or even the user's connection bandwidth. The Linux Virtual Server [LVS01] is an Open Source project aiming to provide a LB cluster software solution on Linux and is a good example of a LB cluster.

High-Availability (HA) clusters improve the availability of a service by having as much redundancy as possible. This is achieved by redundant hardware components within the nodes themselves, but also redundant power and communication channels between the cluster members. This process of adding redundancy to a cluster is often referred to as resolving so called Single Points Of Failure (SPOF). If a hardware component, communication or power channel or even a complete node fails one of the other nodes, components or channels are used in order to continue unaffectedly. If a complete node fails, its resources are taken over by another node. The user usually does not notice this failover at all or at least simply has to logon again in order to establish a new session, but this depends on the session management of the application being used to access the failed resource. The fact that the resources are failed over from one node to another is the reason why an HA cluster is also often referred as a failover cluster. Examples of HA cluster software products for Linux are provided in chapter 3. HA clusters are often enforced by marketing and management in order to guarantee a certain amount of availability of a (cluster) system. The term High-Availability and how it is measured are discussed in detail later on in this chapter.

In practice, it is very common to have hybrids of the three cluster types described above. For instance, it is likely that a Load-Balancing cluster also implements a High-Availability cluster in order to guarantee a certain availability of the nodes the user requests are being forwarded to. In addition, if a LB cluster only has one front-end accepting user requests the complete cluster becomes unavailable if that machine goes down. Therefore, the front-end is often also implemented as a HA cluster.

As this thesis' goal is to implement a HA cluster solution for IDS on Linux, the term cluster is used in the following chapters in order to refer to a HA cluster. The other cluster types are not considered any further as the field of clusters is way to complex to cover it here completely and it would go beyond the scope of this thesis.

1.3. High-Availability (HA)

Business depends on computers more than ever. The world knows this at least since events like the Y2K issue [BBC01], the terrorist attacks on the World Trade Center in New York, USA on September 11th 2001 [S11N01] or any other disaster of the last years. Events like this make clear, why having redundant components in a computer system is quite important, at least in the business world: one of the components can fail and bring down an entire system. Depending on the importance of that failed system and the field a company operates in, the resulting downtime can cost thousands or even millions of U.S. dollars. These are only the possible direct costs though, not to mention the possible indirect costs such as decreased customer satisfaction or even image loss. With (nearly) everybody realizing this, the term High-Availability (HA) and HA clusters in general became more and more popular.

While the term HA (or sometimes simply availability) originates probably from the marketing and management areas, the term fault tolerance describes the same, but is much older. The term HA is all over advertising campaigns and product brochures nowadays. Nevertheless, certain customers (mostly in the financial sector) won't buy a clustering product if it is labeled high-available instead of fault tolerant. So it actually is a marketing related issue which of the both terms to use. After deciding

how to name it, the next questions that arise are: What does high-available, respectively fault tolerant, really mean? And when is a system actually determined as high-available? Can HA be measured?

Defining the term HA is not that easy at all (just like the term cluster) as different opinions on this topic exist and especially when a system is entitled as high-available. In fact, there is no fix definition of HA. This becomes even clearer when regarding the entry for HA at the dictionary of the Storage Networking Industry Association (SNIA):

“The ability of a system to perform its function continuously (without interruption) for a significantly longer period of time than the reliabilities of its individual components would suggest. High availability is most often achieved through failure tolerance. High availability is not an easily quantifiable term. Both the bounds of a system that is called highly available and the degree to which its availability is extraordinary must be clearly understood on a case-by-case basis.” [SNIA01]

The fact that even dictionaries, like the one from SNIA, define HA rather vaguely instead of giving a fixed definition as usual shows the difficulties in defining the term HA. Evan Marcus and Hal Stern define High-Availability as:

“High availability, n. A level of system availability implied by a design that is expected to meet or exceed the business requirements for which the system is implemented.” [MaSte01, p. 34]

The above quoted definition by Evan Marcus and Hal Stern is the definition used by this thesis when speaking of HA.

So, the “high” in HA varies from case to case. This still leaves the question open if and how HA, or just availability, can be measured. There exist several methods of measuring and determining the availability of a system. The most common one is calculating the so called “nines” of a system, which is a representation of system availability (uptime) over a certain time period, most often a year, in percent. Table 1 shows such a representation [MaSte01, p. 10].

Table 1: Availability Representation in Percent

Percentage Uptime	Percentage Downtime	Downtime per Year	Downtime per Week
98%	2%	7.3 days	3 hours, 22 minutes
99%	1%	3.65 days	1 hour, 41 minutes
99,8%	0.2%	17 hours, 30 minutes	20 minutes, 10 seconds
99.9%	0.1%	8 hours, 45 minutes	10 minutes, 5 seconds
99.99%	0.01%	52.5 minutes	1 minute
99.999%	0.001%	5.25 minutes	6 seconds
99.9999% (“six 9s”)	0.0001%	31.5 seconds	0.6 seconds

Often a project leader or manager demands 100% of availability for a system. This is almost impossible or at least very hard to achieve as adding redundant components to the system in order to “get another nine” is very expensive. So, after presenting them a list of needed system components and their costs, most project leaders and managers will reconsider demanding 100% of availability!

That the above table and the “nines” in general are not the universal template for calculating a system’s availability has several reasons, best shown by an example calculation: A single machine may have an availability of 99%, but in a system consisting of six machines each with an availability of 99% the overall availability of that system is calculated as 0.99 to the power of six, which equals to approximately 0.9415 respectively 94.15%.

This simple example shows that the more components are added to a system, the more complex it gets and the lower the overall availability of that system gets.

Another issue with these calculations is the fact that the availability of networks is hard to determine, especially when the parts of a system are spread across and need to communicate over the Internet. Furthermore, only downtime in general is counted.

It is not distinguished between acceptable downtime, meaning the time period while the system is not or hardly needed and unacceptable downtime, meaning the time period during office hours.

The above assumes the percentages of system availability of the single components are given, but how is the availability of a system calculated? Evan Marcus and Hal Stern answer this question by the following simple formula [MaSte01, p. 17]:

$$A = \frac{MTBF}{MTBF + MTTR}$$

- A is the degree of availability in percent
- MTBF is the Mean Time Between Failures
- MTTR is the Maximum Time To Repair
- While the MTTR approaches zero, A increases towards 100 percent
- While MTBF increases, MTTR has less impact on A

Here a short example to demonstrate how this formula applies: A system with a MTBF of 100,000 hours and a MTTR of six minutes has an availability of 100,000 / 100,000 + 0.1, resulting in 99.9999%. This is a quite impressive availability level, but its implausibility becomes clear when looking at the MTBF of 100,000 (in words: one hundred thousand) a little bit closer: this means that the system is allowed to have an overall downtime of six minutes in 11.4 years which is quite unrealistic!

So the fastest repair time is (almost) useless, if the accepted time between component failures is set to high. All of the above shows clearly how true the main statement of Evan Marcus and Hal Stern's book is and how well it applies to HA and HA clusters in general:

"High availability cannot be achieved by merely installing failover software and walking away" [MaSte01, p. 5]

The field of High-Availability (clusters) is a complex one and a lot of planning, designing, reconsidering and maintaining has to be done in order to reliably increase a system's availability with a lasting effect. In other words: High-Availability is a process, not just a product!

While Pfister devotes a separate chapter to High-Availability [Pfi01, chapter 12], Evan Marcus and Hal Stern devote a complete book, titled "Blueprints for High Availability", to the subject of HA [MaSte01]. Both are highly used as reference for this chapter and they are good resources for further reading on the subjects of this chapter.

2. IBM Informix Dynamic Server (IDS)

2.1. General Overview

IBM Informix Dynamic Server (IDS) is one of the two primary database systems IBM offers. IDS runs on various operating systems including Linux, Microsoft Windows Server, and major Unix derivates such as Sun Microsystems' Solaris, IBM's AIX and Hewlett-Packard's HP-UX. Based on this, IDS therefore supports several hardware platforms such as PowerPC, Intel x68 and Sparc [IBM01]. This allows IDS easy integration in heterogeneous server environments.

Like all major database systems on the market, IDS databases can be accessed via APIs for several programming languages including C, COBOL, Java (via JDBC), PHP, Python, Perl and even Ruby [IBM02 and Ruby01].

IDS is valued by its customers for its strong online transaction processing (OLTP) capabilities and thereby providing a high amount of stability, availability and scalability. A lot of the tasks that database administrators (DBAs) have to perform in order to maintain other database systems from the market are not necessary with IDS as IDS handles a lot of them by itself. This can reduce the total costs of ownership (TCO) significantly [IBM03]. IBM's strategy is to use IDS in the fields of high-volume OLTP, mass deployments, environments with limited DBA resources, environments that require a high amount of uptime without interaction of a DBA and similar scenarios [IXZ01] whereas IBM's other database system DB2 is more specialized on data warehousing and working together with SAP solutions [IBM04]. A survey stating IDS customers having five hundred days of uptime or even more and other statistical facts about IDS was published on the IDS portal Informix Zone [IXZ02].

There are quite some points that differentiate IDS from other database systems, here are the major ones:

- **Multi-Threading instead of Processes**

  Instead of starting a new process for each request the database system needs to process, IDS has several so called virtual processors (VPs) each specialized on a specific task. When a request arrives at the VP that listens for incoming connections, the request is split into several sub-tasks such as data-read, joins, grouping, sorting. These sub-tasks are then handled by threads generated within the specific VP. This allows a high-performance parallel processing of the different sub-tasks instead of producing unnecessary overhead by creating several new processes whereas threads share and get their memory from the VP. As an example, would one thousand users access a specific database system at the same time then database systems not using multi-threading would create around one thousand separate processes, one for each user request. IDS on the other hand would create around one thousand threads which share memory among each other and create much less overhead. That is why IDS is very popular in the field of heavy OLTP and well-known for its robustness and scalability. An important fact to consider in this context is the much higher CPU and memory usage of database systems that use processes instead of multi-threading. So a multi-threading-based database system does not only increase performance, but also saves quite some costs for hardware.

- **User Defined Routines (UDR) and User Defined Types (UDT)**

  A feature not all database systems on the market offer and if they do, not as detailed as IDS does it, are the so called UDRs and UDTs. UDRs let developers extend IDS by routines that apply to their individual needs and that are then integrated and handled by the database system. The same holds for UDTs, except that they describe special field types developers can define for data their individual software solutions use. A famous example is geographical coordinates, such as GPS coordinates, and according routines that calculate distances between two points for instance. Those routines are then usable within SQL-statements as any other built-in routine of the database system and therefore result in less data being returned to the requesting application which will then in return have to process less data and respond to the user's request more quickly. Without such functions the application would need to

retrieve much more data and do the distance calculations and comparisons by itself. The described UDRs and UDTs are often referred to as the technique called IBM Informix Data Blades.

- **Footprint and Embedability**

  The footprint is the disk space a fresh installed software application needs. In the case of IDS this is around three hundred megabytes (MB) after installing with the default options and removing the unneeded setup files afterwards. This size can be decreased by telling the setup utility to not install specific parts, the Global Language Support (GLS) package for instance. The footprint of a functional IDS system can therefore shrink to a size of about fifty MB. Most other database systems on the market have a much bigger footprint. This enables IDS to be easily embedded into systems that require a small but powerful database system.

- **Fragmentation**

  The term fragmentation refers to the process of distributing data across several storage disks according to constraints defined by the DBA. As an example, a system could have a table containing orders distributed over four storage disks, one for each quarter. A DBA could now tell IDS to distribute the data over the four storage disks according to the quarter the order was booked in. The first storage disk would then hold only orders from the first quarter, the second disk for the second quarter and so on. When a user now requests all orders of the third quarter of a specific year, the according SQL-statement would then be processed by a thread which would compare the request parts to the fragmentation constraints and apply them. In the described example case, IDS would only make a read request to the third disk as it is the only one containing data for the third quarter. Would a request for data of the first and third quarter arrive, IDS would have three threads: two for reading the according data of the two disks and another one for joining the fetched data together and returning it to the user. This example scenario with the described example SQL-statement requesting only data for the third quarter is illustrated in Figure 4. (The figure is inspired by a figure of the whitepaper “Comparing IDS 10.0 and Oracle 10g” by Jacques Roy [Roy01]).

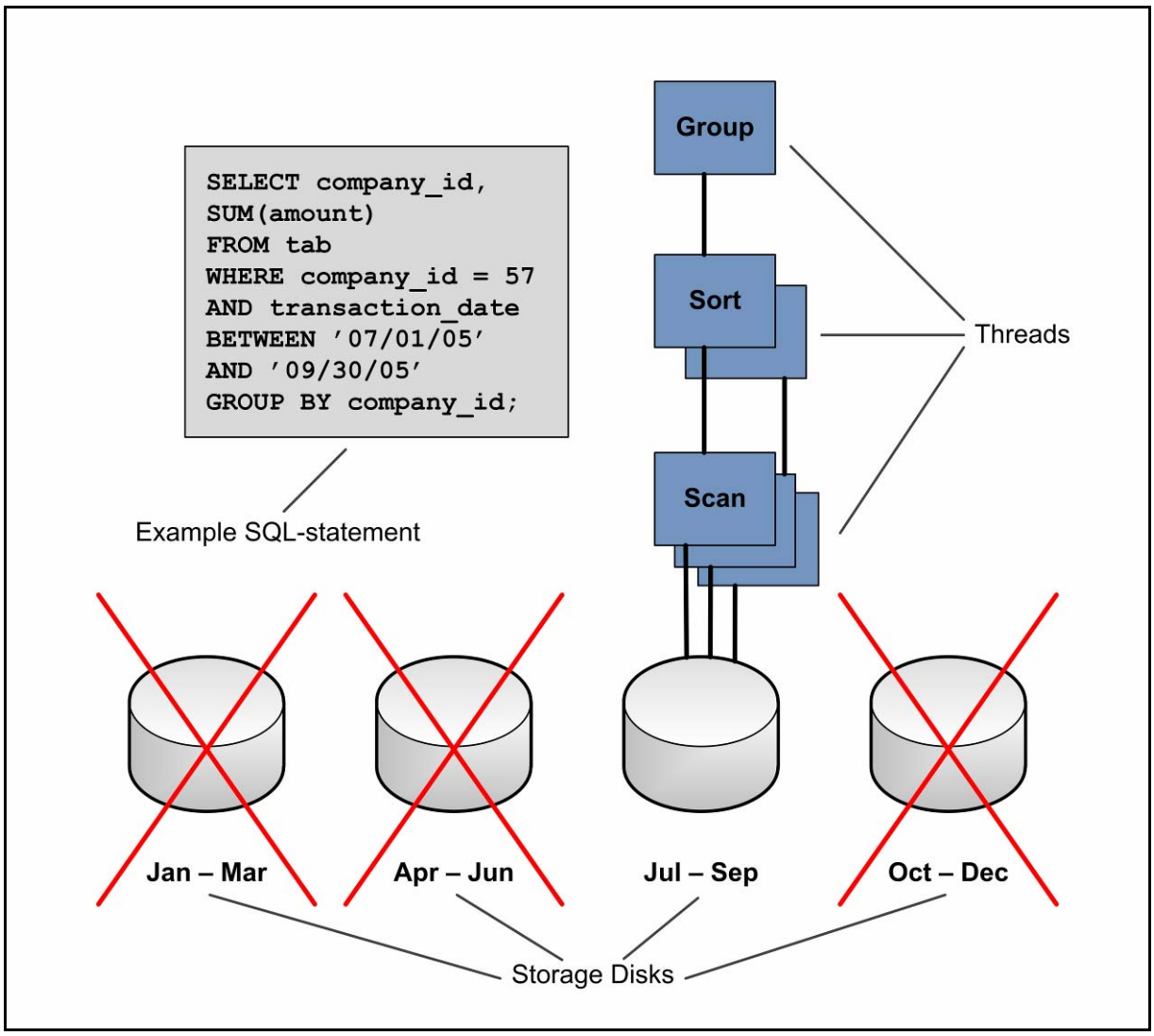


Figure 4: IDS Fragmentation

The term IDS refers in this thesis to the current version 11 code-name “Cheetah”, unless a specific version number is given.

Further and much more detailed information on IDS and its architecture and features are available in the Information Center for IDS 10 [IBM05] and in the IBM Redbook “Informix Dynamic Server V10…Extended Functionality for Modern Business” by Chuck Ballard and his Co-Authors [IBM06]. In addition, the IBM press release for IDS 11 gives a good overview on the new features of IDS 11 [IBM07].

2.2. HA Solutions for IDS

2.2.1. IDS High-Availability Data Replication (HDR)

IDS offers a built-in High-Availability solution, High Availability Data Replication (HDR). As it is a built-in solution, it provides functionalities similar to an HA cluster software, but on application level.

HDR is not the only HA feature IDS offers. By executing different user requests in subtasks that are handled by threads the different requests do not influence each other and if one fails the other ones still continue their processing. IDS also allows a DBA to define mirrored chunks in the database system. Besides providing protection against disk failure (the mirrored chunks are located on two separate storage disks), this also increases performance. This performance gain is performed by telling the SQL optimizer to redirect write requests to the primary chunk and read requests to the secondary (mirrored) chunk. As the two chunks are located on different storage disks this boosts I/O operations a lot. Another HA feature of IDS are hot backups. With hot backups IDS can create a backup of a running instance without stopping the instance and while still answering user requests. It is even possible to let another instance concurrently read the backup while the backup is still being created by the first instance.

HDR is the only built-in HA feature that offers a service failover, though. The basic principle behind HDR is quite simple: A primary machine runs an IDS instance configured in primary mode. This primary takes care of all user requests. The secondary is a second machine running an IDS instance in secondary (standby) mode. Both machines have local storage disks to hold the database data. While the secondary waits and listens to the primary's status, the primary replicates the changes made to his data periodically to the secondary. If the primary fails and does not answer to the secondary's status requests anymore, the secondary switches to primary mode and answers the incoming user requests as the former primary did before failing. In addition, it is possible to configure the secondary for read-only mode

in order to increase performance. The advantage of HDR is that it is easy to set up and no additional cluster software is needed.

Figure 5 illustrates in detail how HDR works. Changes made to the data set on the primary are logged in the logical log buffer. From there they are written to the primary's storage disk and to the HDR buffer on the primary. Depending on the setting of the configuration parameter DRINTERVAL the entries in the primary's HDR buffer are sent to the secondary's reception buffer immediately (synchronous replication mode) or periodically (asynchronous replication mode). The secondary handles the received log entries as if it was restoring a backup, because in this case no special HDR commit functions are needed and the already built-in backup and restore methods can be used. In synchronous replication mode, a transaction committed on the primary is regarded successful as soon as the secondary acknowledged the reception of the according logical log entry. In asynchronous replication mode, the state of a committed transaction on the primary is determined independently from the secondary. If HDR is configured in synchronous mode and the secondary is unavailable, the primary tries to contact the secondary four times. When there is still no answer after the fourth attempt, the secondary is accounted to be dead, the error is logged and the primary continues as if HDR was not configured at all.

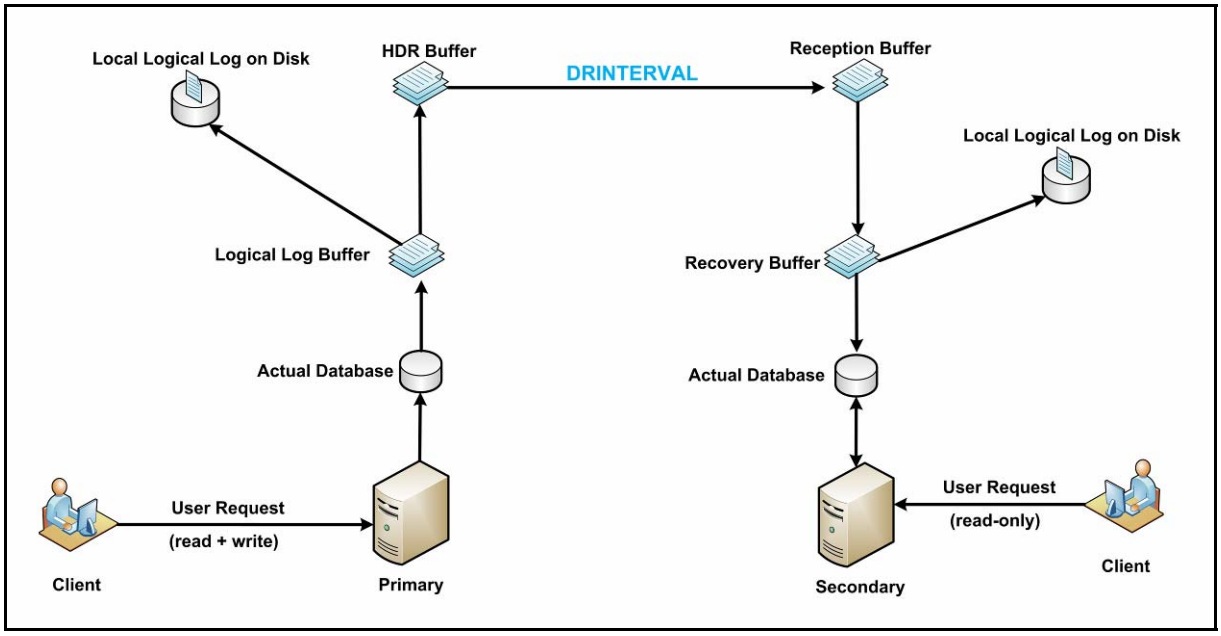


Figure 5: How HDR works

This in detail explanation shows some of the possible drawbacks when using HDR. Here is a list of the major disadvantages of using HDR:

- Binary Large Objects (BLOBs), i.e. images or any other binary data stored in the database, are not replicated and have to be replicated separately.

- Primary and secondary have to run the same operating system, different releases (not versions) are allowed though. For instance, running AIX 5.2 on the primary and AIX 5.3 on the secondary is possible whereas running AIX 5.2 on the primary and AIX 4.x on the secondary is not possible. Running different operating systems (as an example: AIX on the primary and Linux on the secondary) is not allowed at all.

- The IDS releases (not versions) have to be the same on primary and secondary. As an example, running IDS 9.4 on the primary and IDS 9.3 on the secondary would not work, they would have to be both running IDS 9.4 or 9.3.

- The sizes and mount points of the storage disks have to be the same on both machines. In comparison, when using a cluster software the storage is abstracted to the database system and handled by the cluster software as any other resource.

- The client applications connecting to a specific database (and server) need to know the primary and the secondary in order to know who to contact if the primary is not available anymore. An alternative would be to add an abstraction layer by setting up an intelligent DNS server that forwards the user requests depending on the primary's availability. In that case the clients would just know a virtual domain name that gets mapped to the primary or the secondary, if the primary is down. In comparison, a cluster software can handle virtual IP addresses as a normal cluster resource that gets failed over if the node fails on which the IP is currently assigned to.

- IDS needs to run on the primary and secondary. This means that two IDS licenses have to be purchased which increases the software costs of an HA solution using HDR. Depending on the cluster software this can still be cheaper than purchasing a proprietary cluster software license though

- Replication produces higher bandwidth usage and can result in the need of further network hardware. Due to the redundancy recommendations of any cluster software manual, this is a general drawback of HA solutions though.

Besides HDR, IDS offers a feature called Enterprise Replication (ER). With ER specified parts of the data set are replicated asynchronously to more than one peer. It is even possible to configure ER to allow updates being made on and replicated to any of the ER cluster members. ER however does not offer a failover process like HDR does. That is why ER is not regarded as an HA feature. Nevertheless, HDR and ER can be combined to offer a performance HA solution. This chapter provides only a short overview on HDR and ER, as the topics of HDR and ER are explained in much greater detail in the IBM Redbook “Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution” by Chuck Ballard and his Co-Authors [IBM08] and the IDS manual “IBM Informix Dynamic Server Enterprise Replication Guide” [IBM09].

2.2.2. IDSagent for Sun Cluster 3.x on Sun Solaris

The IDSagent is a resource agent for Sun Cluster 3.x, a proprietary High-Availability cluster software solution for Sun Solaris. Sun Solaris is a Unix operating system developed by Sun Microsystems. Sun’s solution offers integration for resources such as applications, shared storage disks or shared IP addresses as well of course. In fact, Sun’s cluster software requires shared disks that are accessible by all nodes, also called multi-host disks. In addition, Sun Cluster has an API which enables vendors and users of any application to integrate it into a high-available and scalable cluster.

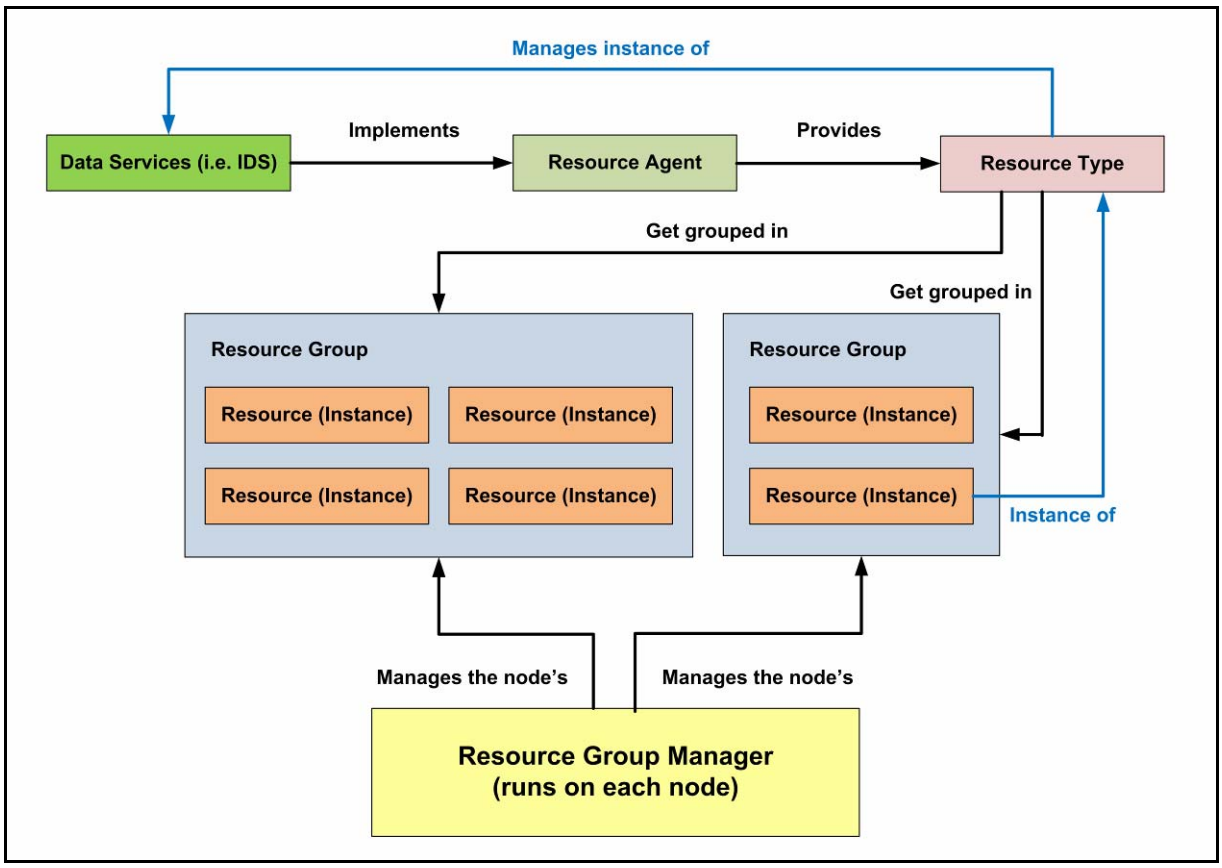


Figure 6: Components Diagram of Sun Cluster 3.x

Figure 6 illustrates how the Sun Cluster software functions: The Resource Group Manager (RGM) is a daemon that needs to run on each node of the cluster. It controls and manages the different member resource groups of the cluster. Thereby, each resource must be assigned to a resource group. Resource groups can be only brought on- or offline at once. A resource represents an individual instance of a data service, a third-party application that's supposed to run on a HA cluster. In the case of the IDSagent the third-party is an IDS database server. In fact, the IDSagent is nothing other than a resource type which inherits its methods to start, stop and monitor an IDS instance to the individual resource instances within the HA cluster. These individual resource instances are then summarized to resource groups, as mentioned already before. This makes it possible to run several IDS instances, even of several IDS release versions, within the same resource group. This does not only hold for IDS though, but for any resource type defined for Sun Cluster 3.x.

Sun's HA cluster solution has some drawbacks:

- Proprietary source code → no individual changes possible or expensive
- Requires shared storage disks → expensive hardware, more complex
- Requires Sun Solaris → does not run on any other operating system
- Expensive license fees, compared to free Open Source alternatives

More details on Sun Cluster 3.x and the IDSagent can be found in the datasheet for Sun Cluster 3.x [Sun01] and especially the whitepaper on how to set up an IDS HA cluster based on Sun Cluster 3.x [IBM10]. Another good resource is to view the Sun Cluster online documentation center offered directly by Sun Microsystems [Sun02].

3. HA Cluster Software Products for Linux

3.1. Overview on HA Cluster Software for Linux

In order to find a suitable HA clustering software for Linux extensive research is necessary. As there are far too many HA cluster software products for Linux on the market, this chapter will concentrate on the six most significant products that were found during the research phase of the thesis. These eight are: OpenSSI, LinuxHA.net, Linux-HA aka Heartbeat, Ultra Monkey, IBM High Availability Cluster Multiprocessing (HACMP), HP Serviceguard for Linux, Veritas Cluster Server and Red Hat Cluster Suite (RHCS).

IBM HACMP, HP Serviceguard for Linux, Veritas Cluster Server and RHCS are commercial products with quite expensive license fees [IBM11, HP01, Ver01, RH01]. This does not fully comply with the thesis' non-functional requirement demanding the solution to be as cheap as possible or even based on Open Source software [Appendix A.1]. As the components of RHCS are published under an Open Source license, the source codes of the different components are freely available, but the major part of the contained programs only supports the Red Hat Linux distribution. In addition, a solution that depends on a product the user would have to compile by himself (and even recompile it every time a new release is published) is not desirable. That is why IBM HACMP, HP Serviceguard for Linux, Veritas Cluster Server and RHCS are not considered any further for a more detailed analysis. The four remaining products are published under Open Source and therefore comply with at least one of the non-functional requirements. Ultra Monkey is a bundle of several Open Source clustering software products. As it's HA features depend on Linux-HA [UM01], Ultra Monkey is not considered any further as well. A description and more detailed analysis of the three remaining candidates follow.

OpenSSI

Name	OpenSSI
Related Websites	http://www.openssi.org [OSSI01] http://wiki.openssi.org [OSSI02] http://openssi-webview.sourceforge.net [OSSI3]
Latest release	April 4th 2005: OpenSSI 1.2.2 (stable) August 2nd 2006: OpenSSI 1.9.2 (development) (checked on June 4th 2007)
License	GNU GPL, Version 2
Description	OpenSSI is an Open Source project that has the goal to provide clustering software that combines the concepts of several software products together. It thereby concentrates on the fields of availability, scalability and manageability. The most significant features of OpenSSI are: single root and init, support for cluster filesystems, distributed lock manager, single process space, process migration via load balancing and load leveling and a single space management.
Supported platforms	Fedora Core 2, Debian Sarge, Red Hat 9
Installation process	A distribution must only be installed on the init node as the other nodes will use that node as a boot server. The boot manager has to be GRUB, as LILO is not supported anymore in current OpenSSI releases. If any of the other nodes needs a different network driver, this has to be loaded on the init node by editing /etc/modules.conf. Then the install script of the OpenSSI package asks a few questions and

	installs the according packages. If special features, such as Cluster Virtual IP, running a NFS server or enabling load-balancing, are to be enabled the user is directed to the according readme files. After the install script finishes, a reboot is required to apply the changes OpenSSI made to the system (and kernel).
Configuration process	Adding new nodes is done by booting them over network with the init node as their boot server. This is handled by the DHCP server running on the init node. This will only work if the network interfaces of the nodes support network booting. Running the openssi-config-node tool will ask a few questions before finally adding and configuring the new node. Adding new services is done by using the distribution's init tools such as chkconfig and the according init scripts in the directories for the seven run levels. The documentation does not make fully clear how to define a service as a failover resource within the HA cluster.
Community	There are four mailing lists (announcements, users, developers, and CVS commits) with low to moderate traffic on them. In the project's wiki about thirty team members are listed. No other signs of a user or developer community can be found on the project's website. Google returns 95.300 results when searching for "OpenSSI".
Documentation	There exist several documents for the 1.2 stable release and the 1.9 development release. Mainly they explain some of the concepts and how to install the software on the

	three supported Linux distributions (→ Supported platforms above). All in all, the project's documentation is moderate and the concepts are not explained good enough to understand how to integrate own applications into a failover cluster.
Popularity	As the project's official sponsor is Hewlett-Packard, expectations for the project's outcome are high. These expectations are not really satisfied considering the small number of search results Google returns and the rather small community of OpenSSI (→ Community above). In conclusion, the project has a moderate popularity.
Max. number of nodes	125
Integrating new applications & Conclusion	As mentioned above, the documentation does not really explain how to integrate an application into the cluster for failover. The parts of the documentation that talk vaguely of the concepts of OpenSSI seem to imply that applications are integrated by placing init start and stop scripts for the specific application into the according run level directories. Own init scripts will have to be written for applications that do not come with init scripts. The disadvantage of this approach is that the cluster software depends on the init tools of Linux and the specialties that arise when using different distributions. As a result, the cluster software has to be ported to any distribution that should be supported. An abstraction to handle an integrated application as a resource that is migrated from one node to another is

	missing. Instead OpenSSI concentrates on processes and migrating processes. This binds OpenSSI more or less to the Linux operating system and its specialties for treating processes.

LinuxHA.net

Name	LinuxHA.net
Related Websites	http://www.linuxha.net [LHAnet01]
Latest release	May 12th 2007: 1.2.6 (checked on June 4th 2007)
License	GNU GPL, Version 2
Description	LinuxHA.net is an Open Source project developed by Simon Edwards in his spare time. It provides failover features for several applications and thereby depends on data being replicated by DRBD (DRBD is introduced chapter 4). Except for the DRBD part, the cluster software operates completely in the user space and is written in Perl. The goal of the project is to provide an inexpensive cluster solution for Linux using replication via DRBD instead of expensive specialized storage hardware.
Supported platforms	Slackware, CentOS, Suse, Fedora Core, Mandrake
Installation process	As packages for the supported platforms exist, the installation process there is quite easy as everything is managed by the respective package manager. For all other Linux distributions, a tar ball based installation is

	possible and not very complicated using the installation guide from the documentation section of the project website. LVM, SSH, Perl and DRBD have to be installed and configured in advance though. As LVM, SSH and Perl are contained in the basis of the major distributions their installation and configuration is done in the most cases already during setup of the distribution. The major distributions come with packages for DRBD which makes the installation process quite easy. The DRBD configuration process includes loading the DRBD kernel modules, adapting the default configuration file and doing an initial sync of the two DRBD peers. The difficulty of this process can be accounted as medium. LinuxHA.net must be installed and configured on all nodes of the HA cluster.
Configuration process	The configuration of LinuxHA.net is done via several XML files in /etc/cluster. With the help of the step-by-step guide from the project documentation setting up a basic cluster with failover for an IP address and out-of-the-box supported services, such as Apache Webserver or a MySQL database server, is of low to medium difficulty.
Community	Except for the author's name, Simon Edwards, no other names are mentioned directly on the project's website. Only the name Colleen Romero is mentioned as one of the authors of the step-by-step guide for LinuxHA.net. Google returns 1.150 results for the search term "LinuxHA.net". The forums link on the project

	website is dead. The community activity therefore seems quite low.
Documentation	The documentation of LinuxHA.net is available as several PDF files as download. The PDF files themselves are well structured and written comprehensible. The level of documentation is therefore good and could serve as a good inspiration for some of the various Open Source projects with poor documentation.
Popularity	Considering the low number of search results Google returns and the fact that only one single person is developing LinuxHA.net, the popularity is regarded as low.
Max. number of nodes	2
Integrating new applications & Conclusion	Each application that is managed by the cluster has a separate folder in /etc/cluster/<application>/. In this folder, the configuration files lie that tell the cluster how to start and stop the application, besides other general parameters such as dependencies for other applications or which IP address to assign to the application within the cluster. So it seems that, similar to OpenSSI, LinuxHA.net depends on start and stop scripts provided by the application itself. Therefore, if none are provided they have to be written before being able to integrate the application into the LinuxHA.net cluster. Similar to OpenSSI applications still remain applications in LinuxHA.net, no abstraction to resources is done, although it does not concentrate as much on single processes as OpenSSI does it.

Linux-HA aka Heartbeat

Name	Linux-HA aka Heartbeat
Related Websites	http://www.linux-ha.org [LHA01] http://wiki.linux-ha.org [LHA02]
Latest release	January 11th 2007: 2.0.8 (checked on June 4th 2007)
License	GNU GPL and LGPL
Description	Linux-HA aka Heartbeat is an Open Source project founded by Alan Robertson in 1999. The project's goal is to provide a HA clustering software for Linux and other platforms. The project implements the Open Cluster Framework (OCF) standard.
Supported platforms	Linux distributions: Suse, Mandriva, Debian, Ubuntu, Red Flag and Gentoo. Other platforms include: FreeBSD, OpenBSD, Sun Solaris and Mac OS X
Installation process	The Linux-HA website offers RPMs and tarballs for download. The RPMs can be easily installed via the RPM package manager. For the other supported Linux distributions appropriate packages exist in their different repositories. All in all the installation process is quite easy. Linux-HA must be installed and configured on all nodes of the HA cluster, meaning the configuration files are identical on all nodes as well.
Configuration process	For a basic configuration the default configuration file at /etc/ha.d/ha.cf can be adapted. When using the configuration mode of version 1 the resources are defined in

	/etc/ha.d/haresources. This is very simple and examples can be found on the project website of the documentation that came with the software. Using the configuration mode of version 2 implies to activate the Cluster Resource Manager (CRM) in /etc/ha.d/ha.cf and configure the cluster and its resources by using the GUI or passing XML snippets to the cibadmin tool. The GUI still has some bugs and not all features provided by Heartbeat version 2 are accessible via the GUI.
Community	On the project's website eleven developers are listed by name, but regarding the list of recent changes to the website indicates that more than eleven people are actively working on the project. The front page of the website states that Heartbeat is being downloaded over one hundred times per day which is quite some traffic. There are three mailing lists (announcement, main, development) with moderate to high traffic on them. Additionally, the RSS feed from the source code repository, the security mailing list or the bug tracking tool can be used. An IRC channel with moderate traffic on it has been set up as well. Google returns 351.000 results when searching for the term "Linux-HA". All in all the community around Linux-HA is quite big and still growing which lets assume that the project will not be closed within the next time.
Documentation	The project website is in fact the public view of the project's wiki. This means anybody that registers to the wiki can edit the website and its

	several pages explaining the features and functionalities of Heartbeat or giving configuration examples. There are a lot of helpful hints, examples and explanations on the website. The issue is that they are spread over the website instead of bundled into a single manual, like with LinuxHA.net for instance. The two screenshots describing how to install and configure Heartbeat and how to use the GUI are helpful as well. Furthermore, several articles and a talk captured as video are linked within the website. All in all, the documentation is only moderate because it takes quite some time for finding something specific. A manual bundling all the spread information together would be desirable.
Popularity	Considering the size of the community and the hits that Google returns the popularity is quite good. Reading the list of project friends (IBM, Suse, Intel, tummy.com ltd, SGI and others) supporting the project with resources or even paid staff seems to justify classifying the popularity as good.
Max. number of nodes	Version 1 configuration mode: 2 Version 2 configuration mode: n
Integrating new applications & Conclusion	Integrating new applications into Heartbeat is done by writing a so called resource agent. A resource agent actually can be any program or script as long as it complies with the OCF standard. This means returning the right exit status codes when a certain case occurs and providing the needed methods such as start, stop, status, monitor and such. In reality, all

	resource agents shipped with Heartbeat are shell scripts. In those shell scripts the binaries or scripts of the application that is to be integrated are called. The resource agent then returns an exit status code depending on the exit status codes the application itself returned. In Heartbeat there is an abstraction of the integrated applications. Everything that is managed by the HA cluster is regarded as a resource. This can be applications, IP addresses, mounted file systems and many more. Heartbeat gets shipped with a large number of resource agents already though, so depending on the resource to be included it might be possible that no special resource agent has to be written. New resource agents are being constantly developed which also represents the community activity and its interest in the Linux-HA project.

3.2. Choosing an HA Clustering Product

Out of the three above analyzed remaining candidates, one is to be chosen now as a basis for the later implementation. The detailed analysis serves as a foundation and justification for the decision. In order to compare the three products directly, ten criteria are defined, which are in detail:

- Product is up-to-date (→ Up-to-date)
- Product's License – how does it comply with the non-functional requirements
- Platform availability
- Difficulty of installation (→ Installation)
- Difficulty of configuration (→ Configuration)
- Community activity
- Quality of Documentation (→ Documentation)
- Project popularity on the market
- Maximum number of possible nodes (→ Max. no. of nodes)
- Difficulty of integrating new applications (→ Integration difficulty)

The terms in parentheses indicate the synonym used for criteria names too long to be used in the table later on. For each of these criteria a candidate obtains an integer score between 1 and 5 whereby 1 is worst and 5 is best. In the following table the different scores of each candidate are listed and compared to the others. Finally, the total scores are summarized for each candidate and compared to the others. The candidate with the highest total score is chosen for the implementation of this thesis.

Table 2 represents the results of the detailed analysis: OpenSSI is on third, LinuxHA.net on second and Linux-HA aka Heartbeat on first place. The HA cluster software product that will be used for the implementation part of this thesis will be Linux-HA. The total scores in the table show that Linux-HA with a total score of forty-one has a clear advance over the two other candidates with total scores of thirty and twenty-eight. Although the documentation of LinuxHA.net is way better than the documentation of Linux-HA, Linux-HA has the clear advantage of being further

developed and having a greater user and developer community, besides also having a good popularity not alone by having several big companies as sponsors. Another advantage of Linux-HA is its abstraction. Everything, be it an IP address, a file system or an application, is regarded as a resource. Resources can be summarized in groups. This not only makes the administration of the HA cluster easier, but also improves the portability of Linux-HA nit only across Linux distributions but also other operating systems such as Sun Solaris or Mac OS X. The latter creates the possibility to have a single HA solution for multiple operating systems instead of using two separate cluster software products, one for Sun Solaris and one for Linux. Once the IDS resource agent for Linux-HA is written, it could therefore maybe even replace the proprietary HA solution of the IDSagent for Sun Cluster (introduced in chapter 2.2.2).

Table 2: Comparison Table of HA Cluster Software Products for Linux

Criterion	OpenSSI	LinuxHA.net	Linux-HA
Up-to-date	2	3	3
Product's license	5	5	5
Platform availability	3	4	5
Installation	3	4	4
Configuration	3	4	4
Community activity	2	1	4
Documentation	2	4	3
Project Popularity	2	1	4
Max. no. of nodes	4	1	5
Integration difficulty	2	3	4
TOTAL SCORE:	28	30	41

4. Distributed Replicated Block Device (DRBD)

DRBD stands for Distributed Replicated Block Device and is a special kind of block device for the Linux operating system. DRBD was developed by Philipp Reisner and Lars Ellenberg and is a registered trademark of LINBIT, an IT service company in Austria [LINBIT01]. It is published as Open Source software under the GPL Version 2 [GPL01].

DRBD provides data replication over a network connection and makes the data of a two-node cluster highly available. Each DRBD block device can hereby be in primary or secondary state. The device in primary state forwards all write requests to the local hard disk and to the secondary device on the peer cluster node. This leads to an up-to-date data set on both nodes, which can be regarded as a sort of network RAID-1, a completely mirrored disk. If the primary device becomes unavailable, the secondary should become the new primary device in order to provide ongoing service. This can be accomplished either manually by executing a few shell commands, or a cluster software can handle this failover automatically. When the former primary device comes back up again, it becomes the new secondary and synchronizes the changes made to the data set in the meantime with the new primary. Figure 7 represents a typical two-node cluster running DRBD. The primary device is on the active cluster node and the secondary on the standby node. The data replication is done over a dedicated crossover Ethernet connection between the two cluster nodes in order to minimize bandwidth usage on the external network connection and optimize throughput of the replication process.

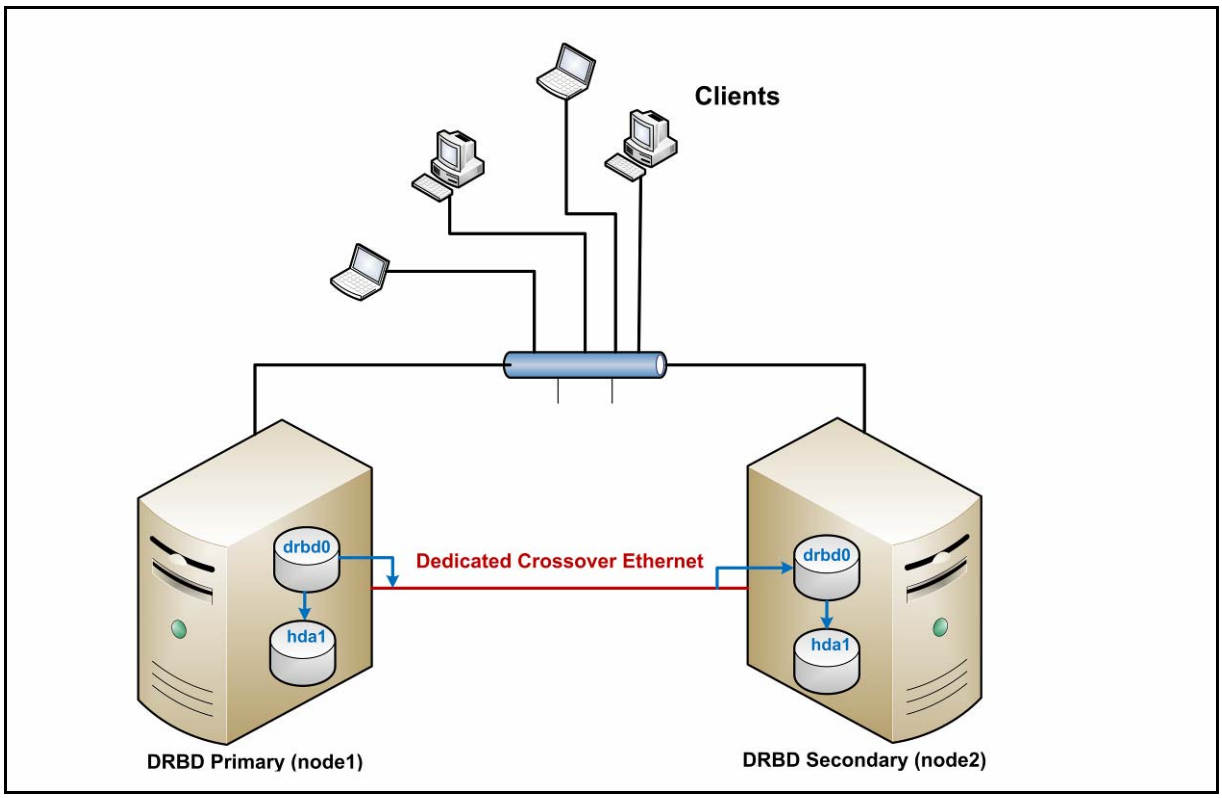


Figure 7: DRBD Two-Node Cluster

So DRBD is a block device that operates between the Linux operating system issuing write commands and the actual physical hard disk and the network interface. Data is then copied to the physical hard disk and to the network interface which transmits it to the secondary device on the peer cluster node. This is an inexpensive alternative to shared storage systems which in most cases only large enterprises can afford. Simply replicating the data between two nodes also produces less overhead and access times compared to a shared storage system which has to transmit the data each time it is requested. Another point is that DRBD is much easier to set up and configure than a shared storage system. These are the reasons why DRBD is used for the test failover cluster system in the development process of this thesis, which will be presented in chapter 6.

When DRBD runs in the standard primary/secondary mode, only the primary can mount the device holding the data. It is also possible to run DRBD in primary/primary mode when using a cluster file system on the hard disks. This allows both nodes to access the data simultaneously, but this case will not be covered by this thesis. These and other more detailed facts about DRBD are well covered on the DRBD project website [DRBD01] and in a sub section of the Linux-HA project website [LHA03]. These resources also provide tutorials on how to obtain, install and configure DRBD. In addition, the article “DRBD in a Heartbeat” by Pedro Pla describes how to set up a failover cluster based on DRBD and Linux-HA version 1 [Pla01].

5. Linux-HA

The Linux-HA aka Heartbeat Open Source project is already introduced in the overview on HA cluster software products for Linux in chapter 3.1. Therefore this chapter skips the general overview on Linux-HA and directly dives into the technical components and their functioning. One point should be restated here again though: In order to function properly, Heartbeat needs to be installed and configured on all cluster nodes, meaning that the configuration files are identical on all nodes.

Version 1 of Heartbeat will not be covered in detail, as it has some major drawbacks and it is highly recommended to use version 2 of Heartbeat. A list of the major issues concerning version 1 that led to the development of version 2 follows [LHA04]:

- Limitation to two nodes per cluster
- No monitoring functionalities for resources
- Few or no possibility to define dependencies for resources

Besides the fact of only being able to have a two-node cluster under version 1, not being able to monitor resources is quite an issue. When a process of a database server defined as a resource is killed, Heartbeat version 1 does not notice this and will try to neither restart nor try to failover the database server. This demands an external monitoring software product that is integrated into Heartbeat version 1 which adds complexity to the overall HA solution. Defining dependencies for resources can be also quite important in some constellations, as an example: If in a two-node cluster one node is a stronger machine than the other, the resources are likely desired to run on the stronger machine if it is available. Constraints can guarantee this, but they were not introduced until Heartbeat version 2.

5.1. Heartbeat Version 1 Configuration Mode

A basic configuration of a two-node cluster running on Heartbeat version 1 configuration mode involves editing three configuration files: */etc/ha.d/ha.cf*, */etc/ha.d/haresources* and */etc/ha.d/authkeys*. The file *ha.cf* contains general settings for the cluster, while in the file *haresources* the resources for the cluster are defined and on which node they should run initially. In the file *authkeys* the digital signature method for the messages the cluster members send to each other is configured.

Listing 1: Sample ha.cf Configuration File – Version 1 Configuration Mode

```
use_logd yes
udpport 694
bcast eth0 eth1
node node1 node2
keepalive 2
deadtime 30
warntime 10
initdead 120
auto_failback on
```

Listing 2: Sample haresources Configuration File

```
node1 IPaddr2::192.168.0.254/24/eth0 apache2
```

Listing 3: Sample authkeys Configuration File

```
auth 1
1 sha1 This is just a simple test cluster system! :)
```

Listing 1 shows a sample general cluster configuration (*ha.cf*) telling Heartbeat in the first line to enable the heartbeat log daemon which then writes its log entries to the configured system log file (mostly this is */var/log/messages*). The second and third line cause the heartbeats to be sent as broadcast messages over UDP port 694 via the network interfaces eth0 and eth1. That the HA cluster consists of the two nodes with hostnames node1 and node2 is configured by the fourth line. The fifth line of the sample configuration defines in which intervals heartbeats are sent to the other node (*keepalive*) and the seventh line defines how long a node waits before declaring

another node as dead after not having received heartbeats from it for some time (*deadtime*). The *warntime* parameter tells Heartbeat how long to wait before printing a warning message that heartbeats got lost, but the other node is not considered dead yet. As starting up the Heartbeat daemon and initializing communication between the nodes takes some time, the *initdead* is usually greater than the *deadtime* parameter and defines how long the starting node waits until starting any resources or making assumptions about the other node's state. The last line of the example configuration tells Heartbeat to automatically fail resources back once they have been failed over to the second node and the first node is available again after being not available for some time. If *auto_failback* is set to off, resources would not fail back to the first node once they have been migrated to the second node.

Listing 2 shows a sample resource configuration (*haresources*) assigning two resources to the node with hostname node1. The syntax is to write the node's name at the beginning of the line followed by a list of resources, separated by spaces. In this case the assigned resources are a virtual IP address and an Apache version 2 webserver. Linux-HA comes with various resource agents, so here it is possible to assign other resource types such as file systems, DRBD devices, IBM DB2 instances, and many more. Even self-written resource agents can be addressed here, which is discussed in more detail later on in this chapter. An active-active configuration would be defined by a second line beneath the one of the above example with a different node name.

Listing 3 presents a sample configuration defining which digital signature method the nodes use to communicate with each other (*authkeys*). The first line tells Heartbeat which rule defined in the same file should be used. The second line defines a rule with index 1, the Secure Hash Algorithm (SHA1) [RFC3174] as signature method and the sentence "*This is just a simple test cluster system! :)"* as pass phrase. Other possible algorithms include the Cyclic Redundancy Check (CRC) [Wikip01] and the Message-Digest algorithm 5 (MD5) [RFC1321]. SHA1 is best and MD5 is next best. CRC is only an error-detecting code and adds no security, like SHA1 and MD5 do. The file *authkeys* must have the file permissions 600 [FAQS.org01], meaning that root only has read and write access to the file, otherwise the heartbeat daemon will not start.

Though Heartbeat version 1 and its configuration style (Heartbeat version 2 still supports the version 1 configuration mode) are obsolete they were introduced above. The reason for this is that configuring resources via the *haresources* file is in some ways easier than a version 2 configuration. Therefore it is used by the initial test cluster system during the development process of the resource agent for IDS, the desired final product of this thesis. The initial test cluster and the development process are described in detail in chapter 6. For the validation process a virtual three-node cluster is set up running Heartbeat in version 2 configuration mode. The validation cluster and validation process are described in chapter 7. The rest of this chapter will concentrate on Heartbeat version 2 configuration mode, the new features introduced with Heartbeat version 2 and how to implement resource agents for Heartbeat.

The chapters 6 to 9 of the book “The Linux Enterprise Cluster” by Karl Kopper [Kop01] give a quite detailed overview on Heartbeat version 1, its features and how to configure it. This book is used as a resource for the brief overview given above, besides the various articles to be found on the Linux-HA project website of course [LHA01].

5.2. Heartbeat Version 2 – Features and Configuration

With releasing Heartbeat version 2 a lot of new features were introduced [LHA05] in order to resolve the drawbacks of version 1, mentioned above. A list of the major features introduced with version 2 follows:

- ✓ Unlimited maximum number of nodes → n-node cluster
- ✓ Monitoring features for resources
- ✓ Various constraints for resources and resource groups
- ✓ GUI for configuring, controlling and monitoring the HA cluster
- ✓ Command line tools for configuring, controlling and monitoring the HA cluster
- ✓ OCF compliant resource agents
- ✓ Automatically replicated resource configurations
- ✓ New cloned and master/slave resource types

The *haresources* file is not used in Heartbeat version 2 configuration mode. Instead, resources are defined in the Cluster Information Base (CIB), an XML configuration file (usually) found at */var/lib/heartbeat/crm/cib.xml* on each cluster node. The *ha.cf* and *authkeys* configuration files remain necessary for Heartbeat version 2 to work though. In order to manage the CIB, Heartbeat has a component called the Cluster Resource Manager (CRM) which has to be enabled for a version 2 configuration. Taking the sample configuration files for the version 1 configuration mode above as a basis, the files look in version 2 configuration mode almost the same. The differences are that the CRM is enabled in *ha.cf* and instead of the haresources file the CIB (*cib.xml*) contains general cluster configuration as well as the configuration for the nodes, resources, resource groups and resource constraints. The modified sample *ha.cf* file with the enabled CRM is shown in Listing 4 below.

Listing 4: Sample ha.cf Configuration File – Version 2 Configuration Mode

```
use_logd yes
udpport 694
bcast eth0 eth1
node node1 node2
crm yes
```

The parameters *keepalive*, *deadtime*, *warntime*, *initdead* and *auto_failback* are omitted here in the modified *ha.cf* file as they are configured directly in the CIB.

As mentioned already, the CIB is a configuration file in XML format. If the *ha.cf* file is modified as shown in Listing 4 and the *cib.xml* is not present, Heartbeat will start but print a lot of errors and warnings to the log indicating that it can not connect to the CIB. This is quite logical as the CIB does not even exist yet. There are two possibilities to resolve this:

- Convert the *haresources* file of the version 1 configuration sample configuration above to a version 2 configuration via the haresources2cib.py script [LHA06]. This will create an initial *cib.xml* file.

- Stopping the heartbeat daemon and manually creating an initial *cib.xml* [LHA07].

A minimal *cib.xml* configuration file looks like shown in Listing 5. The CIB is divided into the two major sections *configuration* and *status*. The *status* section is managed by the heartbeat daemon during runtime and contains information about which nodes are online and on which of the nodes the resources are located. This section should never be touched and touching it could cause the heartbeat daemon to fail or act in any other unforeseen way. In the *configuration* section there are four sub-sections: *crm_config*, *nodes*, *resources* and *constraints*. The *crm_config* sub-section contains general cluster configurations such as *auto_failback* (called *resource-stickiness* since Heartbeat version 2) or various timeouts. As the name suggests, nodes are defined and configured in the *nodes* sub-section, analogous for the *resources* sub-section. A sample how the resources sub-section looks like is shown in Listing 6. In the *constraints* sub-section, constraints for the resources can be defined. With the help of these constraints, resources can be bound to a specific node, resources can be started in a special order or resources can be started depending on the status of another node. Individual variables can be even defined and used in the constraints. In addition, resources can be started depending on date and time, as an example a failover to a specific node can be allowed on weekends only as that node is too

heavily used on weekdays. A sample constraints sub-section is presented in Listing 7. The XML Document Type Definition (DTD) of the CIB gives an exact overview of the configuration possibilities. An always up-to-date version of the CIB's DTD can be found at */usr/lib/heartbeat/crm.dtd*. A little older version of the DTD is also present on the Linux-HA website [LHA08]. It is highly discouraged to edit the *cib.xml* manually while Heartbeat is up and running, except for creating an initial CIB it is even discouraged when Heartbeat is not running. Doing so anyway can lead to unexpected as well as critical behavior. Instead, it is recommended to use the shipped GUI or the command line tools. A complete list of the command line tools shipped with Heartbeat version 2 can be found at the Linux-HA website [LHA05].

Listing 5: Sample Initial cib.xml Configuration File

```
<cib>
   <configuration>
     <crm_config/>
     <nodes/>
     <resources/>
     <constraints/>
   </configuration>
   <status/>
</cib>
```

Listing 6: Sample resources Sub-Section of the CIB

```
<resources>

  <primitive id="cIP" class="ocf" type="IPaddr2" provider="heartbeat">
    <instance_attributes>
      <attributes>
        <nvpair id="cIP_ip" name="ip" value="192.168.0.254" />
        <nvpair id="cIP_mask" name="cidr_netmask" value="255.255.255.0" />
        <nvpair id="cIP_nic" name="nic" value="eth0" />
      </attributes>
    </instance_attributes>
  </primitive>

</resources>
```

Listing 7: Sample constraints Sub-Section of the CIB

```
<constraints>

  <rsc_location id="rsc_location_cIP_node1" rsc="cIP">
    <rule id="prefered_location_cIP_node1" score="100">
      <expression attribute="#uname" operation="eq" value="node1" />
    </rule>
  </rsc_location>

  <rsc_location id="rsc_location_cIP_any_node" rsc="cIP">
    <rule id="prefered_location_cIP_any_node" score="50">
      <expression attribute="#uname" operation="ne" value="node1" />
    </rule>
  </rsc_location>

</constraints>
```

There are three resource types since Heartbeat version 2: primitive, master/slave and clones. The sample in Listing 6 defines a resource of type primitive which is a virtual IP address in this case. The lines within the *attributes* section define the parameters for the IP address, the netmask and the network interface to assign the virtual IP address to. The resource type primitive is the normal case while master/slave and clones are special resource types. To give a brief description, clones are resources that run on several nodes at the same time. Master/Slave resources are a subset of the clones and only run two instances (on different machines) of the given resource. In addition, master/slave resource instances can either have the state master or slave. This comes in handy when configuring a two-node HA cluster based on DRBD, for instance. As clones [LHA09] and master/slave [LHA10] resources are not discussed any further in this thesis, more details about them can be found on the Linux-HA website.

The sample constraints defined in Listing 7 apply to the sample virtual IP address defined in Listing 6. Both constraints are location constraints adding a score to the quorum score of each node (more about quorum later on in this chapter). By scoring node1 one hundred points and any other node fifty points, it is guaranteed that the resource will run on node1 when available. Besides the constraint type *rsc_location*, there exist two other constraint types: *rsc_colocation* and *rsc_order*. The constraint type *rsc_colocation* is used to tell a resource to run on a node depending on the state

of another resource on that particular node. An example would be to define a constraint telling a filesystem resource to only run on a node where a DRBD resource is running and in master state. The *rsc_order* constraint type is used to define the order in which resources should be started, respectively stopped. An example would be to define a constraint that enforces to start the DRBD resource before mounting the filesystem on that DRBD device.

5.3. Heartbeat Version 2 – STONITH, Quorum and Ping Nodes

When reading the Heartbeat documentation and using Heartbeat, there are several important terms one runs across. Three of them are STONITH, quorum and ping nodes. These three terms are discussed in the following:

STONITH stands for “Shoot The Other Node In The Head” and is a fencing technique used in Heartbeat in order to resolve a so called split-brain condition. In order to explain split-brain and STONITH, here a little example scenario: Assuming a two-node cluster with a single STONITH device attached to both nodes. This means that each node can bring down the power connection of the other node by sending a special command to the attached STONITH device. If all communication channels between the two cluster members are lost, each node will assume the other node as dead and try to take over the cluster resources as they appear unmanaged. This case in which both nodes will try to gain control of the cluster resources at the same time is called split-brain. This split-brain condition can only be resolved by applying a fencing technique. Fencing means here to decide which of the cluster members will gain control of the cluster’s resources and force the other cluster members to release them. Deciding which node should take over the resources is not easy as the nodes cannot communicate with each other and it is unclear which node has still connection to the outer world (network), for instance. In order to avoid any uncertain assumptions, the easiest approach in order to resolve this split-brain condition is to force one of the nodes to really being dead by cutting its power supply. This way of

thinking is based on Dunn's Law of Information: "What you don't know, you don't know – and you can't make it up" [LHA11]. Figure 8 illustrates the above example:

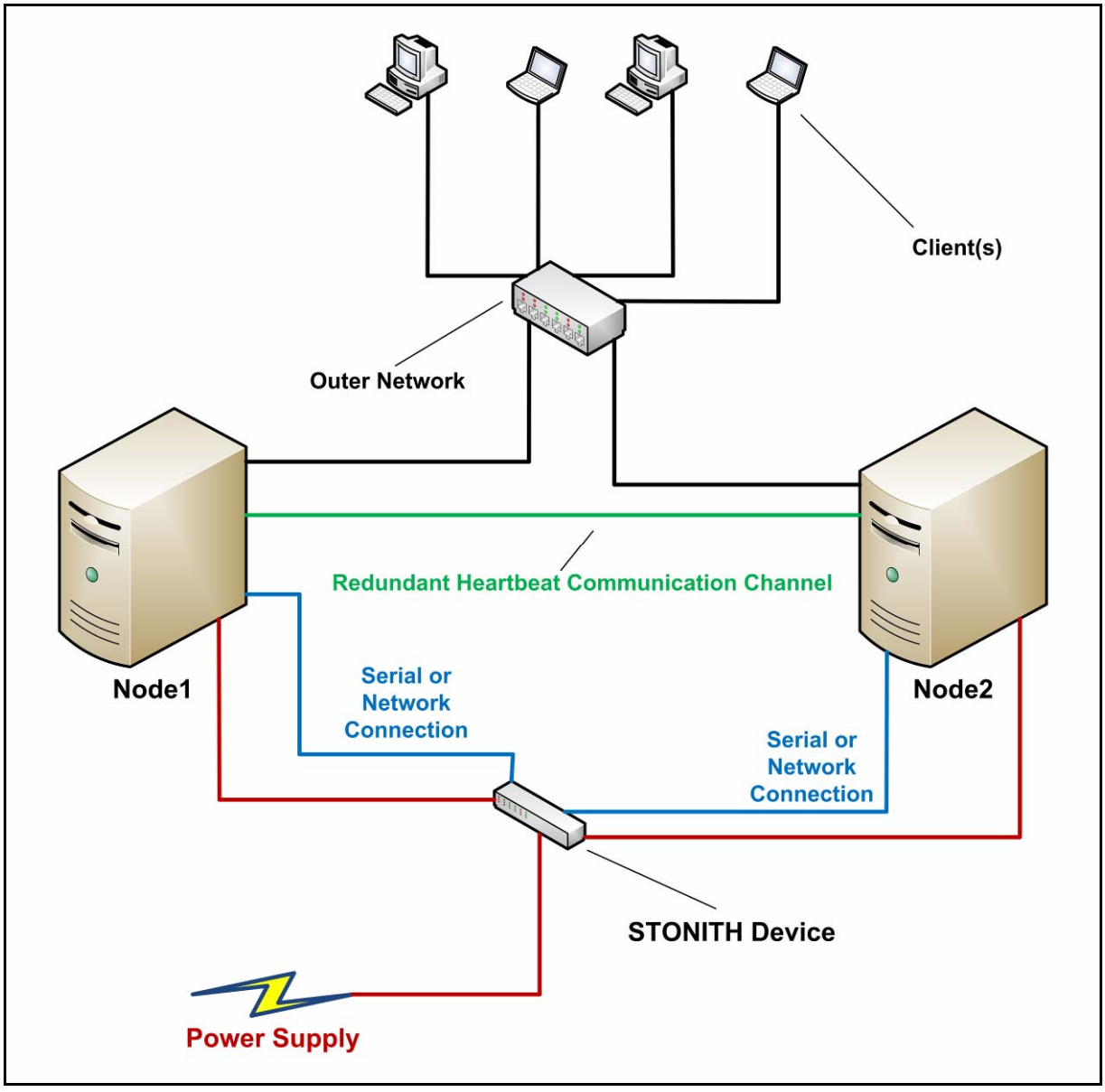


Figure 8: Two-Node Cluster with STONITH Device

The articles on the Linux-HA website on STONITH [LHA12], split-brain [LHA13] and fencing [LHA14] are a good resource for further reading. In addition, the book "The Linux Enterprise Cluster" is also a good resource for further reading on split-brain and STONITH [Kop01, p. 113-114 and chapter 9].

Quorum is a calculation process which determines all sub-clusters there are in a cluster and chooses one of them to be the designated sub-cluster. Only this designated sub-cluster is allowed to operate on the cluster resources. In such a case it is common to say that the cluster has quorum. The best case is when there is only one sub-cluster which means that all cluster nodes are online and operational. If communication between two or more cluster members is lost, several sub-clusters are calculated and the quorum calculation process has to decide which nodes are eligible in taking over the cluster resources. It is quorum that decides which node STONITH should shut down. Quorum is one of the major cluster concepts described on the Linux-HA website [LHA15].

Ping Nodes are pseudo-members of a cluster. They don't have any membership options or even the right of taking over cluster resources. They simply function as a connection reference for the cluster nodes and help the quorum calculation process while defining the designated sub-cluster. Since Heartbeat Version 2, it is possible to define resource location constraints that depend on the number of ping nodes accessible or if any are accessible at all. For instance, a constraint could enforce a resource being stopped and failed over to another node as soon as the node currently holding the resource cannot ping at least two of the defined ping nodes anymore. Ping nodes and how to configure them are explained more detailed on the Linux-HA website [LHA16].

5.4. Heartbeat Version 2 – Components and their Functioning

Heartbeat version 2 consists of several major components which are organized in three levels beneath the init process. Two of them only run on the so called Designated Coordinator (DC) which is the machine that replicates its configuration, better Cluster Information Base (CIB), to all other nodes in the cluster. There has to be always exactly one node running as the DC. Figure 9 shows a process tree view of the major Heartbeat version 2 components. The figure is inspired by the architecture diagram from the Linux-HA website [LHA17].

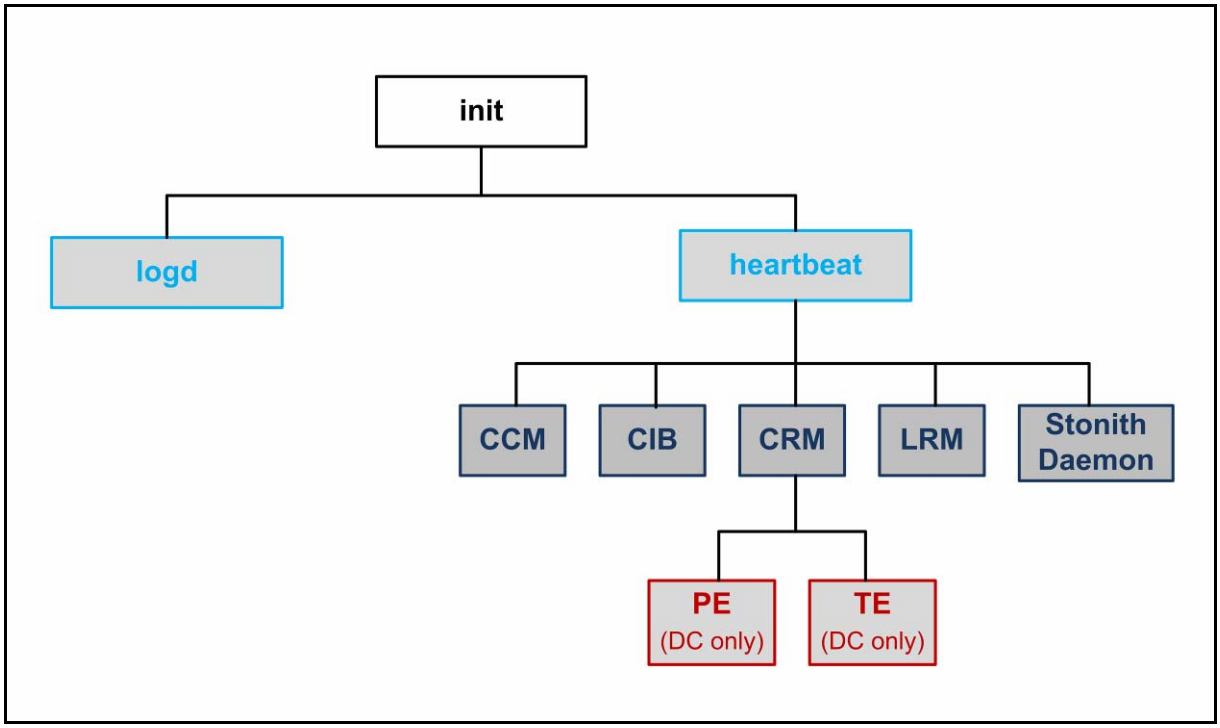


Figure 9: Heartbeat Version 2 – Process Tree View

The two main Heartbeat components are located directly beneath the init process. They are logd and heartbeat. While logd has no children, the heartbeat component has five children: CCM, CIB, CRM, LRM and STONITH daemon. On the machine that is the current DC the CRM has two children: PE and TE. The full name, meaning and purpose of each of these major components are explained in the following:

- **Non-Blocking Logging Daemon (logd)**

  The logging daemon forwards all log messages passed to it either to the system log daemon, a separate log file, or both. The logd can be called by any Heartbeat component. Here the term “Non-Blocking” means that instead of having the component that passes a log message to wait, the logging daemon of Heartbeat takes the message and waits on itself for the log entry to be written while the rest of Heartbeat can continue normal operation.

- **Heartbeat**

  This is the communication layer that all components use to communicate with the other nodes in the cluster. No communication takes place without this layer. In addition, the heartbeat component provides connection information on the members of the cluster.

- **Consensus Cluster Membership (CCM)**

  The CCM takes care of membership issues within the cluster, meaning it interprets messages concerning cluster membership and the connectivity information provided by the heartbeat component. The CCM keeps track which members of the cluster are online and which are offline and passes that information to the CIB and CRM.

- **Cluster Information Base (CIB)**

  This is somewhat the replacement for the *haresources* configuration file as the CIB is a XML file (*cib.xml*) and contains the general cluster configuration, resource configuration with according constraints and a detailed complete status of the cluster and its resources.

- **Cluster Resource Manager (CRM)**

  The CRM is the core of Heartbeat as it decides which resources should run where, and delegates tasks such as to start or stop a specific resource to the different nodes and generate and execute a transition graph from one state to another state.

- **Policy Engine (PE)**

  In order to make decisions the CRM needs a transition graph from the current cluster state to the next state. The Policy Engine generates this transition graph for the CRM. The PE only runs on the DC.

- **Transition Engine (TE)**

  The CRM uses the Transition Engine in order to carry out actions. The TE tries to realize the transition graph generated by the PE and passed by the CRM. The TE only runs on the DC and therefore when a change in configuration or

state of the cluster occurs it is the TE (as a part of the CRM) which informs the other nodes about the changes and gives them orders on how to react to these changes.

- **Local Resource Manager (LRM)**
  Every node has a LRM that receives orders from the TE of the CRM of the current DC. The LRM is the layer between the CRM and the several local resource agent scripts. It handles and performs requests to start, stop or monitor the different local resources of the node it belongs to.

- **STONITH Daemon**
  The STONITH daemon initiates a shutdown or a reset of another node via one of its various STONITH plugins. The LRM therefore has special STONITH resource agent scripts that instruct the STONITH daemon. The STONITH daemon waits for a success or failure exit status code its plugin used for the node shutdown (or reset) and passes that exit status code back to the LRM.

Regarding the sample scenario of a node going down in a cluster helps to better understand how the above described components relate to and interact with each other. The data flow process of such a case is illustrated in Figure 10. As the PE and TE only run on the DC all arrows to or from them are painted in red color in the illustration. A description of each step follows as a numbered list in chronological order of the steps in the illustration:

1) As soon as a node goes down, the heartbeat layer notices the absence of the heartbeats of that node.

2) The CCM periodically checks the connectivity information provided by the heartbeat layer and notices that the connectivity status of the node that went down has changed. It therefore adjusts its state graph of the cluster indicating which members are online and which are offline and informs the CIB and CRM about the changes.

3) The CIB, receiving the status changes from the CCM, updates its *cib.xml* accordingly.

4) The CRM is notified as soon as the CIB is changed.

5) When the CRM notices the changed CIB, it calls the PE in order to have it generate a transition graph from the former state to the new current state (in the CIB).

6) The PE then generates the requested transition graph according to the settings and constraints defined in the CIB. It therefore needs to access the CIB directly.

7) As soon as the PE is done generating the transition graph with an according list of actions to perform (if any needed), it passes them to the CRM.

8) The CRM then passes that transition graph and the list of actions to the TE.

9) The TE then goes through the graph and the list of actions and directs the CRM to inform the LRMs of all the nodes of the cluster that are online.

10) The CRM carries out the directions of the TE and the LRMs on the different nodes then perform the desired actions and return an exit status code which the CRM passes to the TE on the DC.

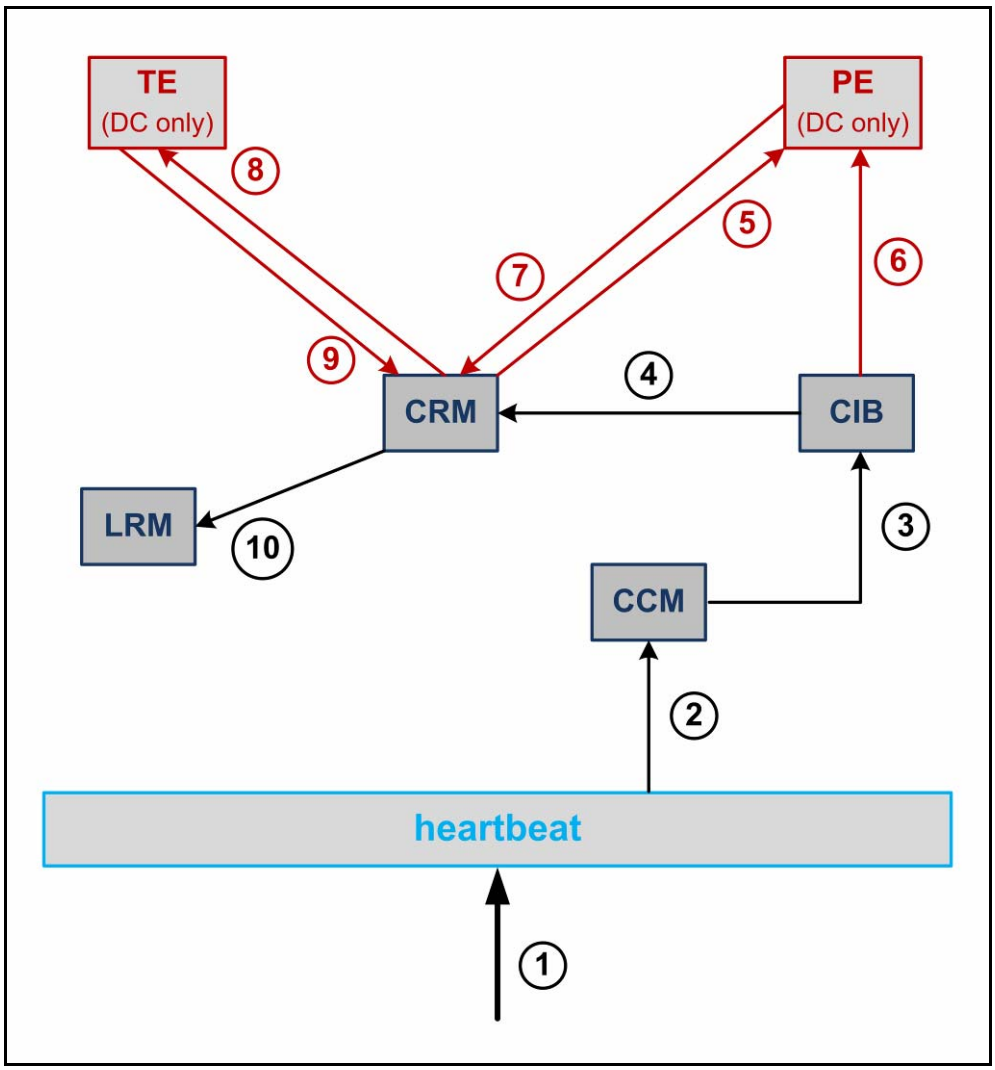


Figure 10: Heartbeat Version 2 – Data Flow

More details on the architecture and concepts behind Heartbeat version 2 can be found on the Linux-HA website, especially the articles about Heartbeat's basic architecture [LHA17] and the new design of version 2 [LHA18]. The rest of this chapter will concentrate on the different resource agent types Heartbeat version 2 offers. In addition, it will have a look at how to develop individual resource agents in order to integrate applications that are not yet supported by Heartbeat version 2. In the case of this thesis, the concrete goal is to decide which resource agent type suits the best for integrating IDS into Heartbeat version 2.

5.5. Resource Agents and their Implementation

As mentioned above, the LRM uses several resource agent scripts in order to start, stop or monitor the various resources of a cluster. Since version 2, Heartbeat supports three types of resource agents [LHA19]:

- Heartbeat Resource Agents
- LSB Resource Agents
- OCF Resource Agents

Deciding which resource agent (RA) type to use for the implementation part of this thesis is rather easy as on the Linux-HA website, as well as in the Linux-HA IRC channel it is highly recommended to write OCF resource agents. In fact, the existent Heartbeat and LSB resource agents are being constantly transformed into OCF resource agents since Heartbeat version 2 is introduced. Heartbeat resource agents are actually LSB init scripts which offer the functions start, stop and status [LHA20]. LSB resource agents are init scripts that are shipped with the distribution or operating system. They comply with the specifications by the Linux Standard Base (LSB) which is a project to develop a set of standards in order to increase the compatibility among Linux distributions [LSB01]. Due to the recommendation to write and use OCF resource agents, the Heartbeat and LSB resource agents are not considered any further in this chapter. More details on LSB resource agents can be found on the Linux-HA website [LHA21]. The fact that the three resource agent types are mostly shell scripts makes them quite similar to each other. The main difference is that the OCF resource agents comply with the Open Cluster Framework (OCF) standards [OCF01]. As Heartbeat version 2 offers functions that implement the OCF standards, developing a resource agent that complies with the OCF standards is quite simplified.

Actually, a RA need not necessarily be a shell script. Any other scripting or programming language could be used as long as it is guaranteed that the script, respectively program, complies with the LSB or even better with the OCF specifications. As all RAs that are shipped with Heartbeat are shell scripts and shell scripts are quite simple to develop, the RA for IDS is a shell script as well. Another good point about shell scripting is the probability of a person using Linux and also

knowing at least the basics of shell scripting is quite high as shell scripts are an essential part of any Linux system. In the following a very simple example of an OCF RA will be shown and explained as a preparation for the development process described in chapter 6.

There are a few rules that have to be taken care of while writing an OCF RA for Heartbeat. These require to return specific exit status codes in certain situations and to support specific methods. The methods a normal OCF RA has to offer are:

- start – starts the resource
- stop – stops the resource
- monitor – determines the status of the resource
- meta-data – returns information about the resource agent in XML format

In addition, if the OCF RA should also support cloned and master/slave resource types the following additional methods have to be supported as well:

- promote – promotes the instance to master state
- demote – demotes the instance to slave state
- notify – used by heartbeat layer to pass notifications directly to the resource

As the desired IDS resource agent does not need to support cloned and master/slave resource types, the actions needed for these are not considered any further here. The method “validate-all” is not needed but it is recommended to have in order to offer Heartbeat the possibility to have the RA check its configuration parameters. Other recommended methods to implement include: “status”, “methods” and “usage”.

The exit status codes the RA should return depend on the current situation and the method invoked. The Functional Requirements Specifications [Appendix A.2] and the Design Specifications [Appendix A.3] for this thesis fully comply with the OCF specifications for RAs for Heartbeat version 2. As these specifications and later on chapter 6 give a quite good overview on how an OCF RA should react in a specific situation, this is not further discussed here and the rest of this chapter deals with a simple example of an OCF RA.

In order to offer the required methods in a shell script a simple *case* statement as shown in Listing 8 suffices. The first line of this example tells the shell where to find the interpreter to use in order to execute the script. The next line then includes the shell functions implementing the OCF specifications. This file is shipped with Heartbeat version 2. As this is a very basic example, the different cases simply print a text to the standard output of the system. At the end, the exit status code of the last executed command is returned to the caller of the script which is probably Heartbeat. In this case this will probably always be equal to 0, indicating success, as the *echo* command normally terminates successfully.

Listing 8: Basic Sample OCF Resource Agent

```
#!/bin/sh
. /usr/lib/heartbeat/ocf-shellfuncs

case "$1" in
      start)
            echo “starting resource”;;

      stop)
            echo “stopping resource”;;

      monitor)
            echo “checking status and functionality”;;

      meta-data)
            echo “printing meta-data”;;

      *)
            echo “undefined method called”;;
esac

exit $?;
```

A more extended example is presented in Listing 9. Compared to the basic sample in Listing 8, this sample illustrates how the different cases perform actions and return an appropriate OCF exit status code depending on the exit status code of the performed task. Furthermore, instead of *echo* commands the *ocf_log* function is called in order to have Heartbeat write the messages to the configured system log instead of standard out. Again at the end, the last exit status code is passed to Heartbeat or the shell if the script was called manually.

Listing 9: Extended Sample OCF Resource Agent

```
#!/bin/sh
. /usr/lib/heartbeat/ocf-shellfuncs

case "$1" in
       start)
              /usr/bin/an/application/startappscript
              rc=$?
              if [ $rc –eq 0 ]; then
                     rc=$OCF_SUCCESS
                     ocf_log info “started resource”
              else
                     rc=$OCF_ERR_GENERIC
                     ocf_log error “error while starting resource!”
              fi;;

       stop)
              /usr/bin/an/application/stopappscript
              rc=$?
              if [ $rc –eq 0 ]; then
                     rc=$OCF_SUCCESS
                     ocf_log info “stopped resource”
              else
                     rc=$OCF_ERR_GENERIC
                     ocf_log error “error while stopping resource!”
              fi;;

       monitor)
              /usr/bin/an/application/monitorappscript
              rc=$?
              if [ $rc –eq 0 ]; then
                     rc=$OCF_SUCCESS
                     ocf_log info “resource is up and working”
              else
                     rc=$OCF_ERR_GENERIC
                     ocf_log error “resource not running or not working”
              fi;;

       meta-data)
              echo $xml_meta_data
              rc=$OCF_SUCCESS;;

       *)
              ocf_log info “undefined method called”;;
esac

exit $rc;
```

This extended example can be adapted and used in order to integrate an application into Heartbeat. Nevertheless, there are still two cases missing that have to be implemented: trying to start the application when it is already running, and trying to stop the application when it is already stopped. Also in order to save space, a placeholder (*$xml_meta_data*) for the XML meta-data is used in Listing 9. It has to be replaced accordingly.

More details on OCF resource agents can be found on the Linux-HA website [LHA22]. As mentioned above, the implemented IDS RA for Heartbeat discussed in detail in chapter 6 relies on shell scripting. Shell scripting is not explained in this thesis, as a lot of good literature explaining it in great detail already exists. Two quite good examples are the book “Classic Shell Scripting” by Arnold Robbins and Nelson H. F. Beebe [RoBe01] and the online manual “Linux Shell Scripting Tutorial” by Vivek G. Gite [Gite01].

Part II – Development and Validation Process

"Walking on water and developing software to specification are easy as long as both are frozen." – Murphy’s Computer Laws [Q2]

"Undetectable errors are infinite in variety, in contrast to detectable errors, which by definition are limited." – Murphy’s Computer Laws [Q2]

"Debugging is at least twice as hard as writing the program in the first place. So if your code is as clever as you can possibly make it, then by definition you're not smart enough to debug it." – Murphy’s Computer Laws [Q2]

6. Implementing the IDS Resource Agent for Linux-HA

6.1. Initial Thoughts and Specifications

As with any software project, one of the first things to do is to specify written requirements and guidelines the final product should follow. The project of this thesis is not an exception here. After an in depth theoretical analysis of clusters in general and all of the other major related topics in Part I, it is appropriate to define non-functional and functional requirements specifications. These provide a first impression on how to approach the development of the desired IDS resource agent (RA). The non-functional requirements specification (NFRS) and function requirements specification (FRS) are attached in Appendix A.1. As each of these attached documents already contains a detailed description, only a few of the major points are picked up in the following:

In the first place, the NFRS define the deadline and license under which to publish the results. It also requires the solution to depend on Open Source products as far as possible. This was a great decision indicator while researching and analyzing cluster software products for Linux in chapter 3. In addition, it specifies the requirement that the final solution should be implemented and validated on SLES10 and RHEL5. All of these points played a big role while deciding which cluster software product to choose and how the development environment, which is introduced in detail later on in this chapter, should look like.

As the NFRS also requests that the implementation of the IDS RA should function in configuration modes for Heartbeat version 1 and version 2, a general analysis of resource agents was made in chapter 5. This analysis shows that since Heartbeat version 2, it is common and highly advised to write and use resource agents that follow the OCF standard. Due to historical reasons how Heartbeat itself developed, OCF RAs are not directly usable in Heartbeat version 1 configuration mode. It is quite common since Heartbeat version 2 to write wrapper scripts for the OCF RAs instead of writing an OCF RA for Heartbeat version 2 configuration mode and writing a

Heartbeat or LSB resource agent for Heartbeat version 1 configuration mode. Such a wrapper script simply prepares and partially processes the passed parameters in order to pass them to the OCF RA. This saves development time (writing one RA, instead of two separate ones) and makes the end product less complex.

Once the decision is made, which resource agent type the final IDS RA will be, the FRS is defined. As the IDS RA is an OCF RA, it has to comply with the OCF standard. This standard describes how a resource agent is seen and used by Heartbeat, meaning which methods it must offer and which exit status codes it is supposed to return depending on the current state it is in while the set of possible states is fixed. Therefore the FRS defines the two states the IDS resource can be in as seen by Heartbeat: either running or not running. In addition, eight use cases are specified describing in detail the eight methods the IDS RA offers: start, stop, status, monitor, validate-all, methods, usage, and meta-data. A detailed description of the mentioned use cases and states (including a state transition table) can be found in the FRS.

The next step after writing the FRS is to make a more detailed technical analysis of the desired IDS RA and to provide detailed technical design decisions. The result of these design decisions is the design specification (DS) in Appendix A.3. In the DS, all the OCF related guidelines indicated in the FRS are specified concretely. As a first approach, the DS shows how the IDS RA interprets and communicates the different states an IDS instance can be in. Thereby, the various states an IDS instance can be in are simplified to three different states: running, undefined and not running. Having this information as a basis, the IDS RA can react according to the OCF guidelines as shown in the detailed state transition graph and table in the DS. The final step in the DS is then to use the previously specified states and state transitions in order to define concrete flow charts for the most complex parts of the IDS RA: main section, validate, start, stop, monitor. The flow charts for other parts (status, usage, methods and meta_data) are either implemented analogously or considerably less complex, which would make it redundant specifying these flow charts in the DS as well.

According to the NFRS, the source code of the IDS RA is well commented in order to make potential further development by other developers as easy as possible. Therefore, most of the decisions and steps described above and later on in this chapter should be easily comprehensible by simply studying the source code of the IDS RA attached on the CD-ROM in Appendix D.

As chapter 5 already stated that the majority of all OCF RAs shipped with Heartbeat are shell scripts, this thesis will not make any exception and implement the IDS RA also as a shell script. So this was a rather easy decision.

6.2. Development Environment

As the NFRS requests the final IDS RA to be usable in Heartbeat version 1 and version 2 configuration modes, the development process is performed on a two-node cluster system in Heartbeat version 1 configuration mode. This makes the setup of the development environment less complex and therefore faster. It enables getting first results quicker than when directly developing on a three-node cluster in Heartbeat version 2 configuration mode. In addition, the data sets between the two cluster nodes are replicated using DRBD (already introduced in chapter 4) instead of setting up a shared storage (which is covered by the validation process in chapter 7 though).

In order to avoid taking the risk of accidentally contaminating IBM's internal company network, the development environment is setup using three computers in a detached network via a separate 3Com 100 Mbit/s Ethernet switch, but not connected to the IBM intranet at all. This means that all communicating is made via a single network and no redundant communication channels exist. In a productive system, these SPOF would have to be resolved of course! Besides a standard keyboard, mouse and monitor, the three machines used for the development test cluster have hardware specifications as described in Table 3.

Table 3: Hardware Specifications of the Development Environment

Component	node1	node2	client
CPU	Intel Pentium IV 2.4 GHz	Intel Pentium III 800 MHz	Intel Pentium III 667 MHz
RAM	1 GB	512 MB	512 MB
Hard Disk Partitions	hda1: 1 GB, swap hda2: 38.9 GB, / hdb1: 5.1 GB, /mnt/hdb1 hdb2: 200 MB, /mnt/hdb2 hdb3: 54.7 GB, /mnt/hdb3	hda1: 1 GB, swap hda2: 13.5 GB, / hda3: 5.2 GB, /mnt/hda3 hda4: 200 MB, /mnt/hda4	hda1: 1 GB, swap hda2: 18 GB, / hdb1: 15 GB, /mnt/hdb1
Network Card	Intel 82540, 100 Mbit/s	3Com, 3c905TC/TX-M, 100 Mbit/s	3Com, 3c905TC/TX-M, 100 Mbit/s
CD-ROM Drive	✓	✓	✓

The network configuration of the development environment looks like the following:

- node1: node1.ibm.com, 192.168.0.1 on eth0
- node2: node2.ibm.com, 192.168.0.2 on eth0
- client machine: client.ibm.com, 192.168.0.3 on eth0
- Virtual cluster IP address: cluster.ibm.com, 192.168.0.4 on eth0

The netmask is: 255.255.255.0

Each of the computers used runs SLES10. The hosts *node1* and *node2* run Heartbeat (set up in version 1 configuration mode) with a DRBD device including an according filesystem mount, an Apache Webserver version 2 instance (simplified as “Apache 2”), a virtual cluster IP address and an IDS instance as cluster resources. Furthermore, the host *client* runs a Network Time Protocol (NTP) server the two cluster members *node1* and *node2* use to synchronize their time in order to avoid potential deadtime issues that could arise otherwise. While *node1* and *node2* build an HA cluster on Heartbeat, the machine called *client* is used to monitor the availability of the cluster IP address, respectively the entire cluster as a whole.

The above information is summarized in Figure 11 in order to visually illustrate the development environment:

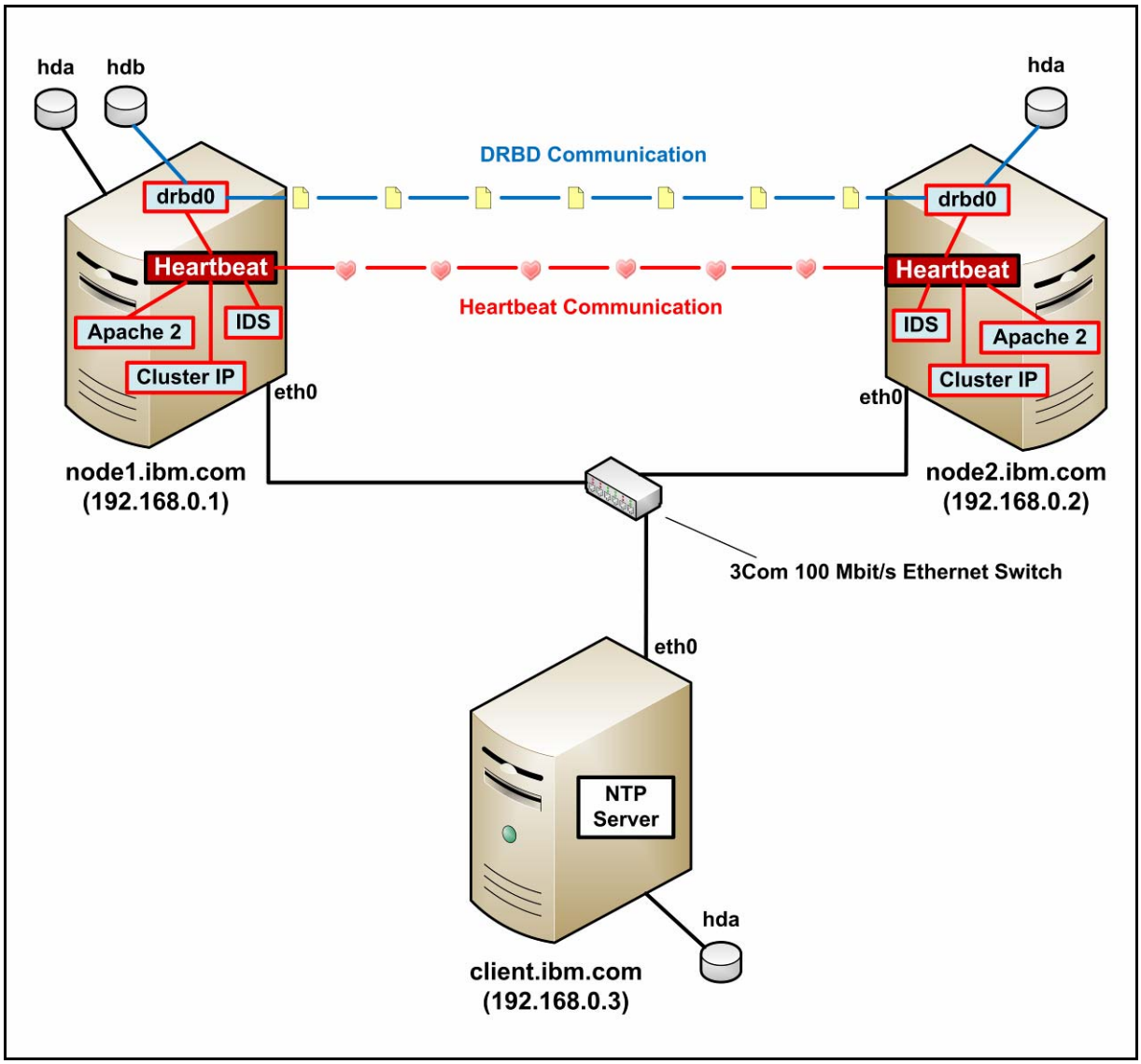


Figure 11: Development Environment Graph

The configuration files of the development test cluster are not discussed here. They are attached on the CD-ROM in Appendix D for documentation purposes though. In addition, there exist tutorials on DRBD and how to set it up and integrate it with Heartbeat on the Linux-HA website [LHA03] and the DRBD website [DRBD02].

6.3. Structuring of the IDS RA in Detail

As the wrapper script for the IDS OCF RA only contains of a header part containing copyright and license information and a main part preparing the passed parameters for the IDS OCF RA, its structure is rather simple. Therefore it will not be discussed here in detail. Instead, this chapter has a closer look on the IDS OCF RA's structure.

It is important to mention that the IDS RA running in Heartbeat version 1 configuration mode does not support the monitor method as Heartbeat version 1 (or Heartbeat version 2 in version 1 configuration mode) itself does not support this. Therefore the configuration parameters "dbname" and "sqltestquery" are not supported when running the IDS RA in Heartbeat version 1 configuration mode.

The IDS OCF RA is logically separated into three major parts: header, function definitions and main section. The header contains, besides copyright and license information, a general description of the IDS RA. As the two parts "header" and "main section" are less complex than the "function definitions" part, they are explained first.

The main section always calls the function *ids_validate* before executing the requested method (i.e. action) passed to the script as one of the parameters. This ensures that any further invoked function can assume the passed configuration parameters as valid and does not have to bother with validating again. Thus the passed configuration parameters are validated every time the IDS RA is called and potential invalid changes to the RA's configuration are detected immediately the next time the RA is called after changing its configuration. Once the main section can be sure the configuration is valid, it calls the function corresponding to the method requested when the RA was called. Finally, the exit status code of the called function is passed on as the exit status code of the RA's requested method. If the main section comes to the conclusion that the configuration is invalid though, it logs an appropriate error message and terminates the script immediately.

The IDS RA configuration parameters are:

- **informixdir** – the directory IDS is installed to
- **informixserver** – the name of the IDS instance Heartbeat should manage
- **onconfig** – the name of the configuration file of the IDS instance
- **dbname** – the name of the database to run the SQL test query on
- **sqltestquery** – the SQL test query to run when monitoring the IDS instance

Besides defining a separate function for each of the eight methods the RA offers, the function definitions also include two helper functions. The ten functions defined in the function definitions therefore are (in chronological order as they appear in the source code):

- **ids_usage**
- **ids_methods**
- **ids_meta_data**
- **ids_log** (first helper function)
- **ids_debug** (second helper function)
- **ids_validate**
- **ids_start**
- **ids_stop**
- **ids_status**
- **ids_monitor**

ids_usage

This function calls the *ids_methods* function in order to retrieve a list of all offered methods and eliminates the line breaks from it. The whitespace separated list of methods is then inserted in a usage description of the RA. Finally the complete usage description is printed to standard out.

ids_methods

This is one of the incomplex functions of the script as it simply prints a list of the methods the IDS RA offers.

ids_meta_data

The *ids_meta_data* function defines the parameters the IDS OCF RA expects and which of them are mandatory or optional. The data is noted in XML format. None of the parameters are marked as required as it is possible to set them as shell variables in advance, more on this in the description for *ids_validate* and the DS. Besides the mentioned parameter settings and a short and long description of the RA, action timeouts for each of the methods the RA offers are a part of the RA's "meta_data" as well. These timeouts tell Heartbeat how long to wait for the IDS RA to respond depending on the method invoked before declaring the method as failed.

ids_log

This function is one of the helper functions. Its responsibility is to take log messages and a type and then to either pass it to the Heartbeat logging daemon or simply print it to standard out via the echo command. The variable "idsdebug" controls whether to process messages of type "info" as well. Messages of type "error" are always processed. The default behavior is to pass only error messages to the Heartbeat logging daemon.

ids_debug

The *ids_debug* function is the second helper function and was created during the development process in order to ease the debugging process when resolving bugs, issues and errors in the script. The function simply compares the current values of the passed configuration parameters and their equivalents in the shell environment

and passes the according info messages to the *ids_log* function. This function is also called when the IDS instance is determined to be in an unexpected or undefined state or any other major error occurred. The debug information printed in this case will certainly help the system administrator in resolving the issue.

ids_validate

The *ids_validate* function is one of the most complex functions of the script. Its responsibility is to analyze the provided configuration parameters and determine whether the IDS RA's configuration is valid or invalid. It assures, if the configuration is determined as valid, that the RA can function properly. If any of the configuration parameters is invalid, appropriate error messages are passed to the *ids_log* function and an according exit status code indicating failure is returned. As mentioned above in the description of *ids_meta_data*, all configuration parameters are optional and therefore not marked as required. In fact, it is possible to define the parameters informixdir, informixserver and onconfig in advance by setting them in the appropriate shell environment variables before calling the IDS RA. The *ids_validate* function will notice this and validate them as if they were passed as normal parameters when calling the RA. It is implemented like this, because IDS itself needs these shell environment variables to be set correctly in order to function properly. So in reality, *ids_validate* takes the parameters that were passed to the RA and sets their according shell environment variables manually if they have not been set already. This leaves the decision whether to centralize these configuration parameters by setting the required shell environment variables in a shell script during system boot or to pass them as parameters to the RA to the system administrator and enhances his flexibility.

ids_start

This function starts the configured IDS instance and returns an according exit status code indicating whether the IDS instance is started successfully or any error occurred during startup. If the start method is invoked when the IDS instance is already

running, the function will simply return an exit status code that indicates that the IDS instance was successfully started. An undefined state leads to an error message and immediate termination of course.

ids_stop

Analogous to the function *ids_start*, the function *ids_stop* stops the configured IDS instance and returns an according exit status code indicating whether the IDS instance is stopped successfully or any error occurred during the shutdown process. If the stop method is invoked when the IDS instance is not running, the function will simply return an exit status code that indicates that the IDS instance was successfully stopped. An undefined state leads to an error message and immediate termination.

ids_status

This function fetches the output of the onstat tool provided by IDS and uses the information to determine the current state of the managed IDS instance. The states are determined as defined in the state definitions in the DS.

ids_monitor

The *ids_monitor* function can be regarded as a sort of extension to *ids_status* as it highly depends on it. It uses *ids_status* to determine the current state of the IDS instance and when the state is considered to be running, an SQL test query is sent to the managed IDS database in order to assure it operates properly. One could also think of this as an enhanced status check of the IDS instance. This function is used by Heartbeat in order to periodically monitor the IDS cluster resource and determine whether it is still functional or not.

Further details on the IDS RA, its structure and functioning can be obtained by analyzing its source code or reading the FRS and DS. The IDS RA source code is attached on the CD-ROM in Appendix D.

6.4. Issues and Decisions during Development

A few issues occurred during the development process and decisions were made in order to resolve them. In the following these issues are listed in chronological order as they appeared:

- As only one set of keyboard, mouse and monitor is available and switching them between the three machines is too time-consuming, the X forwarding option of SSH [Ossh01] is used (using "ssh –X user@host" instead of "ssh user@host"). Therefore it has to be ensured that a SSH server with enabled X forwarding is running on all of the machines. Once this is true, GUI applications running on any of the three machines can be used on the single machine the keyboard, mouse and monitor are currently connected to. The X forwarding option in the SSH server configuration is enabled by default though, so this does not really pose a problem.

- It can happen that the software installer of Yast2 cannot find the CD-ROM though it is inserted in the drive. In such a case it helps to manually create a mountpoint (i.e. */media/cdrom0*), mount the CD-ROM and provide the software installer with the correct URL (i.e. *file:///media/cdrom0*). This is important to know only when installing packages from the CD-ROM discs though.

- The package *drbd* does not have a direct dependency for the package *drbd-kmp-default*. Nevertheless, the latter is needed as it contains the necessary code in order to build the DRBD kernel module.

- DRBD version 8.0 is not yet fully supported by the DRBD OCF resource agent included in Heartbeat. Therefore, DRBD version 0.7 has to be used [LHA23].

- Heartbeat is shipped with SLES10, but in version 2.0.5 which is quite buggy and highly discouraged to use when asking for support on the Linux-HA mailing lists or in the Linux-HA IRC channel [LHA24]. So a manual update to Heartbeat version 2.0.8 is recommended.

- Unfortunately, the DRBD resource agent is still buggy in Heartbeat version 2.0.8 [Gos01]. In order to not loose too much time and being able to start the development of the IDS RA as soon as possible, the development environment runs Heartbeat in version 1 configuration mode using the DRBD Heartbeat resource agent called *drbddisk*. Heartbeat version 1 configuration mode is easier to set up.

- The original version of the *drbddisk* resource agent has to be slightly modified as it contains a bug that keeps the node from becoming the primary node for the DRBD resource again after a failover and failback. Therefore the corrected version of the *drbddisk* resource agent is attached on the CD-ROM in Appendix D.

6.5. First Tests during the Development Process

The IDS RA was periodically tested while developing it. This means each time a new function is added or changed the parts already implemented are completely retested. This assures that changes to one part of the RA do not cause unnoticed side affects on one of the other parts.

In order to test the DRBD setup the DRBD device is mounted on *node1*, a file is created on the device and the network connection between *node1* and *node2* is cut off. On *node2* the device is then mounted and if the DRBD setup is working the file created on *node1* is visible on *node2*.

The Apache Webserver version 2 instance is tested by placing its document root on the mounted DRBD device and creating a simple PHP script which prints the hostname of the machine Apache 2 is currently running on. This means that if a failover from *node1* to *node2* is performed the script prints “node1” as the host it was executed on before the failover and prints “node2” after the failover. The mentioned PHP script is attached, together with the configuration files of the development test cluster, in Appendix D.

Furthermore, the common scenario of forcing a failover from *node1* to *node2* by cutting the network connection of *node1* is run several times during the development process. This scenario also includes a failback from *node2* to *node1* after re-establishing the network connection of *node1* again. After finishing the development process of the IDS RA, a final test of the described scenario was successfully run.

The above described tests during development help to minimize the risk of unnoticed bugs. Nevertheless, they do not replace the need for a separate validation process in order to guarantee a certain level of quality for the IDS RA. In fact, the tests described here are similar to the test cases defined for the validation process. The validation process in chapter 7 will introduce these test cases in detail.

7. Validating the IDS Resource Agent for Linux-HA

7.1. Purpose of the Validation Process

As mentioned at the end of the chapter on the development process, the few tests run during development are undoubtedly helpful but they do not replace a separate validation process. The validation process described in this chapter is supposed to guarantee a basic level of quality for the IDS RA and can be regarded as the project's quality management measures. Quality management or even project management are not covered by this thesis though. The tests run here assure that the IDS RA functions correctly and operates as expected. Of course, not every possible test scenario can be tested here as it would by far exceed the scope of this diploma thesis. Therefore, the list of test cases is limited to eight very common scenarios which serve as a good basis for further test cases. Another reason why not all possible test scenarios of running an IDS instance as a HA cluster resource can be performed here is that they highly depend on the individual infrastructure and the individual Heartbeat cluster configuration. With that it is not appropriate to think of every existing possible customer scenario the number is probably infinite or at least hard to determine. Because the wrapper script included in the IDS RA just prepares variables and then simply calls the IDS OCF RA, the test cases defined in the validation process refer to the IDS OCF RA when using the term IDS RA. After describing the validation environment the tests are run in, the test cases themselves are introduced in more detail in this chapter.

7.2. Validation Environment

Contrary to the development environment and its two-node cluster running Heartbeat in version 1 configuration mode, the validation environment is based on a virtual three-node cluster running Heartbeat in version 2 configuration mode. Heartbeat version 2.1.2 is installed. In addition, as the cluster consists of three members instead of two data replication (via DRBD) is not suitable here. Instead, a shared storage provided by a Network File System (NFS) server is used and mounted as one of the cluster's resources. The NFS server and a NTP server run on a separate machine which also serves the HA cluster as a ping node. The complete environment therefore consists of four machines interconnected via a network switch.

As the hardware used for the development environment is not sufficient, virtualization on a single strong server is used instead. This single server has the following hardware specifications as listed in Table 4.

Table 4: Server Hardware Specifications for the Validation Environment

Component Name	Installed in Server
Model Name	IBM xSeries 235
CPU	2 x Intel Xeon 2,4 GHz with Hyper-Threading
RAM	2 GB
Hard Disks	4 x 37.7 GB Ultra 320 SCSI with integrated RAID-1 mirror resulting in 2 x 37.7 GB usable disks
Network Card	Broadcom NetXtreme BCM5703X Gigabit Ethernet
CD-ROM Drive	✓

The system described above serves as host system for the four virtualized guest systems that form the virtual validation environment. This host system is referenced as simply the host in the rest of the thesis. The host system runs on the Linux distribution Ubuntu, release "Feisty" [Ubuntu01]. The virtualization software used in this environment is the Open Source software VirtualBox [VBox01], developed by the

German IT company innotek [intek01]. This thesis does not cover virtualization in general and a detailed introduction of VirtualBox. The articles on virtualization [VBox02] and VirtualBox itself [VBox03] on the VirtualBox website are good resources for further reading on these topics.

In order to fulfill the requirements of the NFRS, the virtual validation cluster is set up twice: the first time running on SLES10 and the second time running on RHEL5.

The network setup for the validation cluster based on SLES10 looks like the following:

- First node: **sles10-node1, 192.168.15.1 on eth0**
- Second node: **sles10-node2, 192.168.15.2 on eth0**
- Third node: **sles10-node3, 192.168.15.3 on eth0**
- NTP and NFS server: **sles10-san, 192.168.15.254 on eth0**
- Virtual cluster IP address: **sles10-cluster, 192.168.15.253 on eth0**

For the validation cluster running on RHEL5 the following hostnames and IP addresses hold:

- First node: **rhel5-node1, 192.168.15.1 on eth0**
- Second node: **rhel5-node2, 192.168.15.2 on eth0**
- Third node: **rhel5-node3, 192.168.15.3 on eth0**
- NTP and NFS server: **rhel5-san, 192.168.15.254 on eth0**
- Virtual cluster IP address: **rhel5-cluster, 192.168.15.253 on eth0**

Both setups use the netmask: **255.255.255.0**

The HA cluster manages thereby the following cluster resources:

- A virtual IP address for the cluster
- A NFS share used as shared storage
- An IDS database server instance
- A pingd resource in order to check each node's connectivity status
- A meatware STONITH device that requests a manual reboot of a dead node

When ignoring the fact that the validation cluster is simulated on a single machine, the validation environment looks like illustrated in Figure 12:

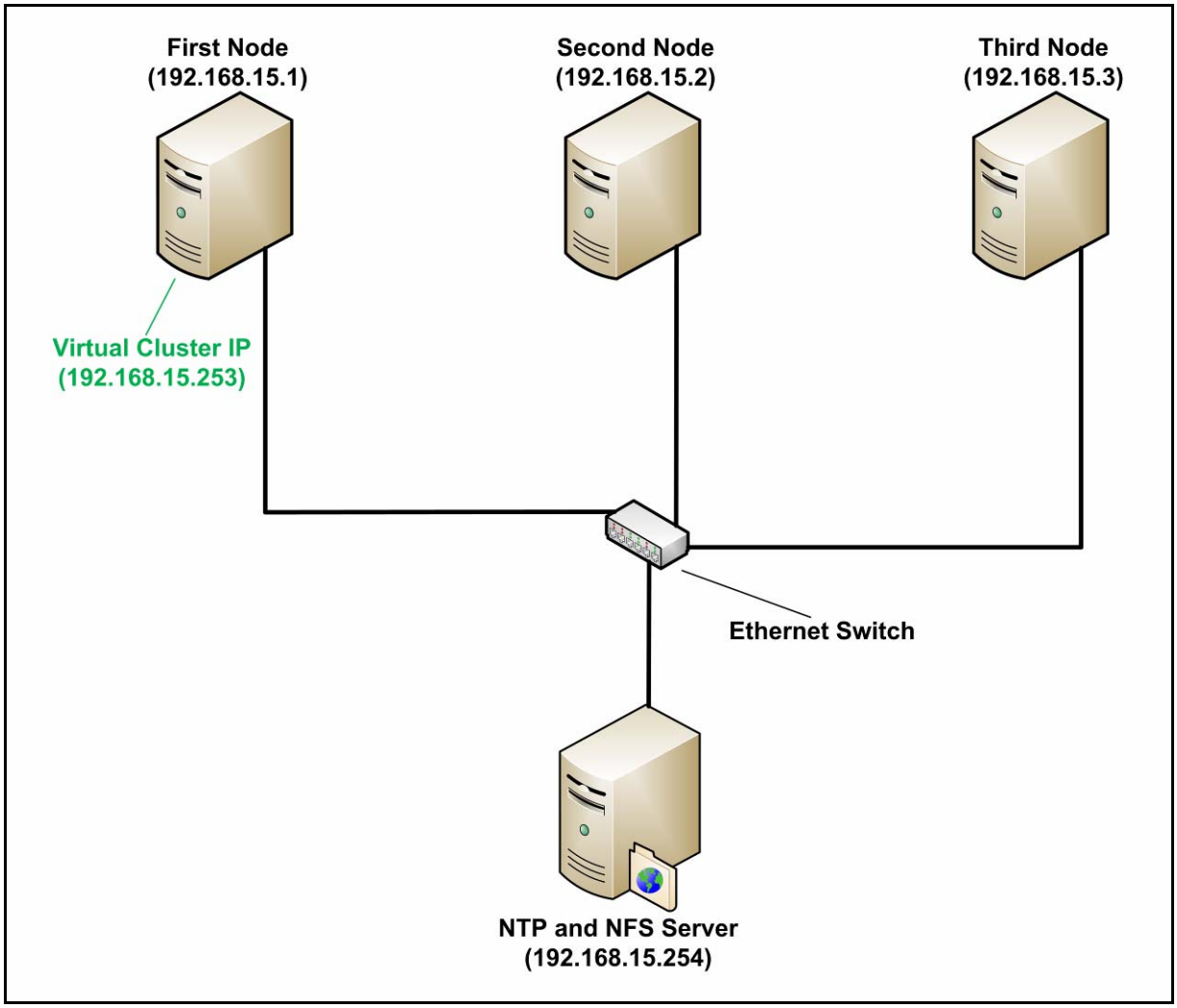


Figure 12: Validation Environment without considering Virtualization

Having a closer look at the validation environment and considering the aspect of virtualization thereby, reveals its complexity, compared to when ignoring virtualization aspects. In reality, there are two physical computer systems and one physical network involved: an IBM ThinkPad T41 (ignoring that IBM's Personal Computing Division was acquired by Lenovo in 2005 [Len01]) and the IBM xSeries 235 specified above interconnected by IBM's intranet. However, simulating a three-node cluster by virtualization software like VirtualBox on that xSeries adds some levels of abstraction, respectively complexity. On the host (the xSeries machine, located in the machine room), VirtualBox is installed and four instances of the program VBoxVRDP run in the background. VRDP stands for VirtualBox Remote Desktop Protocol. It starts a guest system, also referred to as virtual machine (VM), and offers a remote desktop service on a port on the host system. VRDP will not be introduced in detail here as a good and detailed description is already given in the VirtualBox user manual [VBox04, chapter 7.4]. So, all of the machines of the virtual validation cluster are in fact VBoxVRDP processes running on the host system. Each of these processes opens a port in order to connect to the VM with a remote desktop application on the IBM ThinkPad in an office (i.e. outside the machine room). A SSH connection with enabled X forwarding between the IBM ThinkPad and the host system is established and then the program rdesktop is executed for each VM in order to access it. The enabled X forwarding feature in SSH then causes the desktop windows of the VMs, provided by the rdesktop program, not being opened on the host system but on the IBM ThinkPad. As the IBM ThinkPad mainly runs Microsoft Windows, an X server, like Cygwin/X [CygX01], has to be installed and started for this to work. Cygwin/X is not covered in this thesis. This setup has the good side effect of encapsulating the network for the virtual validation cluster in an abstraction layer which assures that it cannot harm the internal company network. The above described validation environment considering the aspect of virtualization is visualized in.

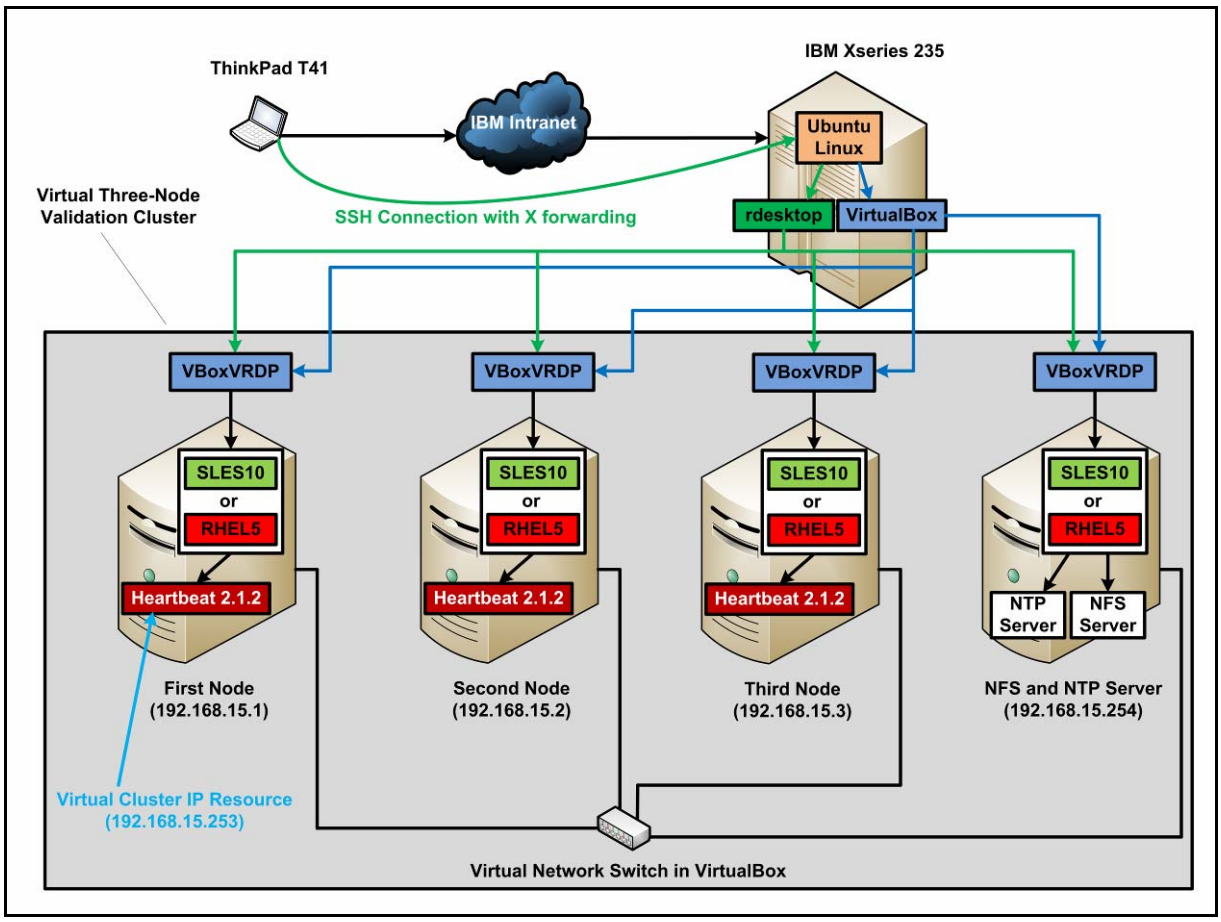


Figure 13: Validation Environment considering Virtualization

The following Table 5 represents the hardware specifications of the IBM ThinkPad used in the validation environment and mentioned above:

Table 5: IBM ThinkPad Hardware Specifications for the Validation Environment

Component Name	Installed in Machine
Model Name	IBM ThinkPad T41
CPU	Intel Pentium M 1,7 GHz
RAM	1 GB
Network Card	Intel PRO/1000 MT Mobile Connection
Display	IBM ThinkPad 1400x1050 LCD panel

How to setup and configure VirtualBox is not covered here as it is not directly related to the IDS RA and the main goal of this thesis. However, these topics are well documented in the documentation section on the VirtualBox website [VBox05] and especially in the VirtualBox user manual [VBox04]. Furthermore, the most common commands used to manage the VMs are enclosed in shell scripts and can be found on the attached CD-ROM in Appendix D. Setting up the three-node validation clusters on SLES10 and RHEL5 and Heartbeat are also not covered in this chapter. Though, an installation guide documenting this (without the virtualization aspects) is attached on the CD-ROM in Appendix D.

7.3. Tests run during the Validation Process

As mentioned above eight test cases were specified for the validation process. Seven of these eight test cases are introduced in the following while the eighth test case has a special role and is presented in a separate section of this chapter.

A test case is noted in a table and consists thereby of the following parts:

- **Test Case ID** – in order to uniquely identify each test case
- **Description** – the current situation and actions being taken
- **Expected result** – the expected result of the test case
- **Output on SLES10** – console and log file output on the SLES10 cluster
- **Output on RHEL5** – console and log file output on the RHEL5 cluster
- **Results on SLES10** – how and if the cluster behaved as expected
- **Results on RHEL5** – how and if the cluster behaved as expected

Short descriptions of the first seven test cases follow:

- The first test case (TC01) checks if the IDS OCF RA passes the test script named “ocftester” which is shipped with Heartbeat. This script verifies OCF compliance.

- In the second test case (TC02) the IDS RA is called manually from the shell. Methods in order to start, stop and monitor the IDS instance are invoked and then the reactions and output of the RA are compared to the output of the onstat tool shipped with IDS. This assures that the RA behaves as expected and does not make false assumptions on the current state of the IDS instance.

- The third test case (TC03) tests whether the IDS resource is restarted when monitoring for this resource is enabled and the IDS process is then killed.

- In the fourth test case (TC04) IDS will be started before starting the Heartbeat software. Heartbeat will try to start the already running IDS instance and is supposed to determine that the IDS resource of the cluster was successfully started.

- The fifth test case (TC05) regards the case when IDS switches to an undefined state, such as the “Single-User mode” for instance. Heartbeat’s monitoring action is supposed to fail and Heartbeat will try to stop the IDS resource. This will fail as well which leads to a failover of the resource if STONITH is configured and one of the other nodes can successfully shut down the node IDS failed on.

- The sixth test case (TC06) assumes a three-node cluster with a configured STONITH device. The node on which the IDS resource is running on is cut off from the cluster by three different variants: manually bring down the network interface, killing all Heartbeat processes and rebooting the machine by simply executing the command “reboot”. In all three cases, the remaining nodes are supposed to declare the failed node as dead and shut it down via STONITH before taking over the IDS resource. This is one of the classic failover scenarios that come to mind at first when thinking of HA clusters.

- In the seventh test case (TC07) the IDS resource is removed from the cluster (whether via the GUI or via the command line tools does not matter). It is expected that Heartbeat will first stop the IDS resource, respectively shutdown the running IDS instance, before removing it from the list of managed cluster resources.

More information on of each of the test cases described above is given in the test cases specification attached in Appendix A.4.

7.4. The IDS Transaction Validation Script (ITVS)

The eighth test case (TC08) has a special role as it is the most complex one and a special script was written for it. The script is called the IDS Transaction Validation Script (ITVS). Considering the script's name, one can assume that the script validates database transactions in IDS before, during and after a failover in an HA cluster environment like the one introduced above. In fact, this is exactly what the script is aimed at and what it does. The phrase "[…] validates database transactions in IDS […]" is a little vague though. That is why a closer look on the transactions the script invokes and especially when they are invoked is given in the following. Before doing that though, the output of calling the script with the parameter "usage" or "help" can provide some more understanding on what the ITVS does. This is shown in Listing 10.

Listing 10: Usage Description of the ITVS

```
sles10-node1:/home/lars# sh itvs.sh usage

    usage: itvs.sh (usage|help|methods|test-before|test-after)

    itvs.sh is the IDS Transaction Validation Script (ITVS)
    and validates if transactions committed on a node
    remain committed after a failover in a High-Availability
    cluster running on Linux-HA aka Heartbeat. The intention of
    this script is therefore to validate the OCF IDS resource agent
    (IDS stands for IBM Informix Dynamic Server).

    This script assumes that IDS is installed and
    the shell environment variables INFORMIXDIR, INFORMIXSERVER,
    ONCONFIG, PATH and LD_LIBRARY_PATH are set appropriately!


    - usage        displays this usage information.

    - help         displays this usage information.

    - methods      simply lists the methods this script can be invoked with.

    - test-before  tells the script to create a test database and start
                   four transactions, of which two are committed before
                   invoking a reboot command in order to force a
                   failover of the current cluster node.

    - test-after   validates if the two transactions committed by 'test-before'
                   remain committed when running this script after failover
                   on the cluster node taking over the IDS resource.

sles10-node1:/home/lars#
```

The script defines four transactions in total. Each of them begins a transaction, creates a sample table and inserts one row of data into that table. The four transactions differ in the point that not all of them are committed. Only the second (t2) and fourth transaction (t4) are ever committed. The transactions t1 and t3 are kept open by sending them into the background and never invoking a “COMMIT WORK;” which is required to commit a transaction. While the two transactions (t1 and t2) are active, a database checkpoint is enforced which writes changed data not only to the log file, but also to the disk. While keeping these two transactions open, two more (t3 and t4) are started. Next, two transactions (t2 and t4) are committed, but t1 and t3 are still kept open. Now a node failover is enforced by simply rebooting the node the IDS resource ran on when starting the transactions. Transactions t2 and t4 are committed after the checkpoint, but before the failover, transactions t1 and t3 remain uncommitted. These four invoked sample transactions cover all major scenarios of the possible ones in this environment. A brief summary of the above follows:

- t1 is opened before the checkpoint and never committed
- t2 is opened before the checkpoint and committed after the checkpoint
- t3 is opened after the checkpoint and never committed
- t4 is opened after the checkpoint and committed before the failover

This is also visualized in Figure 14. As an example, Listing 11 shows the SQL statements of the transaction t4. The other transactions are analogous.

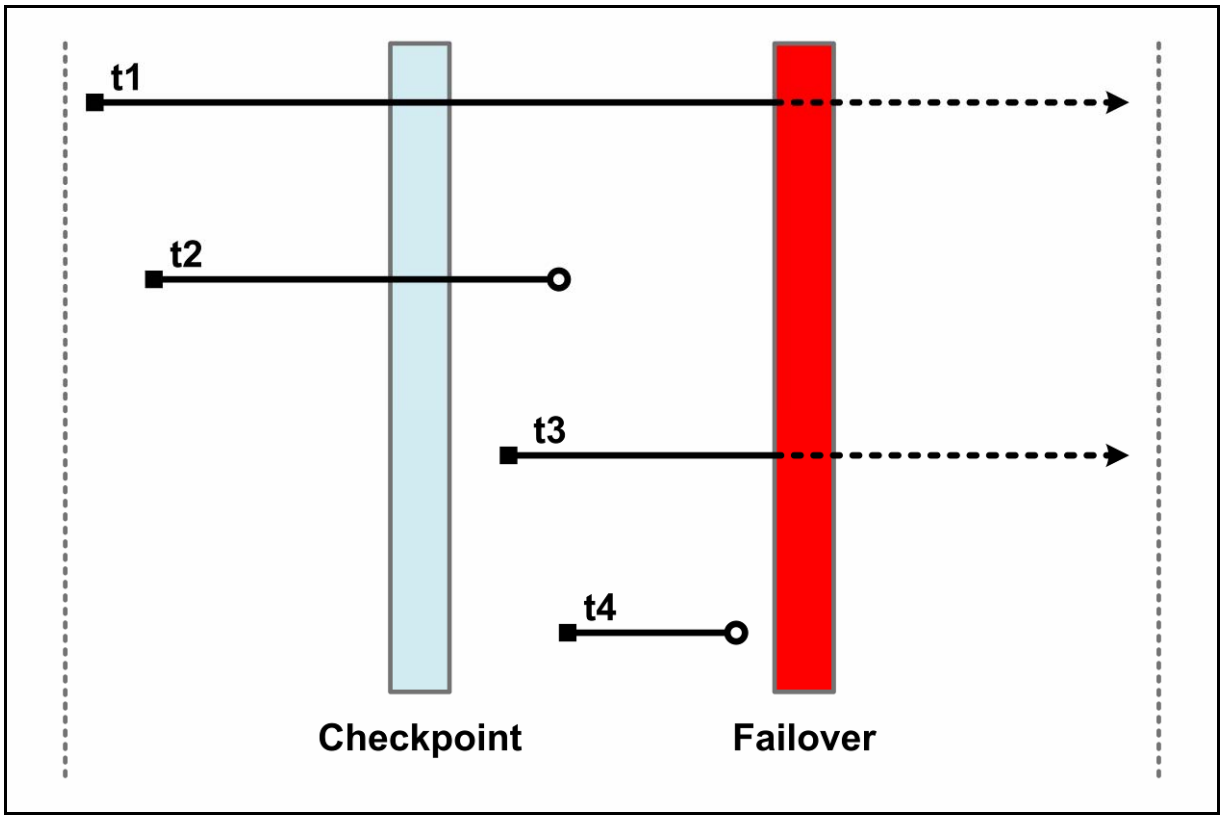


Figure 14: ITVS Transaction Timeline

Listing 11: SQL Statements of the Transaction t4

```
BEGIN WORK;
CREATE TABLE t4 (id SERIAL PRIMARY KEY, text VARCHAR(100));
INSERT INTO t4 (text) VALUES ('test4 test4 test4');
COMMIT WORK;
```

The above described first part of TC08 is done by calling the ITVS with the parameter “test-before”. The output of this first part is shown in Listing 12 which follows:

Listing 12: ITVS Output when successfully passing the Parameter “test-before”

```
sles10-node1:~/ITVS # sh itvs.sh test-before
Processing function istv_test_before
Database created.
Database closed.
Creating test database 'itvs': [ success ]
Executed transaction1 in background...
Sleeping for 10 seconds...
Executed transaction2 in background...
Sleeping for 2 seconds...
Performing IDS checkpoint...
Performing IDS checkpoint: [ success ]
Executed transaction3 in background...
Sleeping for 2 seconds...
Executed transaction4 in background...
Rebooting this cluster node in 20 seconds
(to ensure transaction2 and transaction4 were committed in the meanwhile) in order
to force resource failover of IDS.
Please run 'itvs.sh test-after' after failover on the cluster node the IDS resource
failed over to.
```

While the node the IDS resource ran on is rebooting and the cluster failed over the IDS resource to another node, the script is supposed to be executed a second time on that new node. This time the passed parameter is “test-after” in order to instruct the script to run the second part of TC08. The second part of the test verifies which transactions are committed to the database and which not. If the test is successful only the transactions t2 and t4 are committed. Listing 13 presents how the output of ITVS looks like when the second part of the test is successful:

Listing 13: ITVS Output when successfully passing the Parameter “test-after”

```
sles10-node2:~/ITVS # sh itvs.sh test-after
Processing function itvs_test_after
First Transaction was not committed as expected: [ success ]
Second Transaction was committed as expected: [ success ]
Third Transaction was not committed as expected: [ success ]
Fourth Transaction was committed as expected: [ success ]
SUCCESS: All tests were successful, the resource agent behaves just as expected! :)
Database dropped.
Successfully dropped the test database 'itvs'.
```

More information on the TC08 is given in the test cases specification which is attached in Appendix A.4.

7.5. Validation Test Results

As the document for the test cases specification indicates, the IDS RA passes all specified test cases on SLES10 and on RHEL5. This is summarized in Table 6:

Table 6: Validation Test Results Summarization Table

Test Case ID	Test Result on SLES10	Test Result on RHEL5
TC01	✓	✓
TC02	✓	✓
TC03	✓	✓
TC04	✓	✓
TC05	✓	✓
TC06	✓	✓
TC07	✓	✓
TC08	✓	✓

Thereby the following legend holds:

- ✓ means that the test was passed
- × means that the test was not passed

In conclusion, the entire validation process was completely successful.

7.6. Issues and Decisions during Validation

During the validation process several issues arose which needed to be resolved. The issues thereby occurred on different components of the validation environment, namely these are: VirtualBox, RHEL5, SLES10, Heartbeat and NFS.

A list of the mentioned issues, ordered by severity, follows:

- Heartbeat itself and its STONITH plugins are not well documented which makes it harder to solve issues and misunderstandings while setting up the HA cluster and defining constraints. The Linux-HA website [LHA01], mailing lists and IRC channel [LHA24] sometimes provide advice though.

- Heartbeat release 2.1.2 does not support adding a node to a running cluster without the need of editing the *ha.cf* configuration file and restarting Heartbeat on all nodes in order to obtain the changed node configuration. It is not known yet, when and if this issue will be resolved in one of the future releases.

- Due to the timeouts built in the NFS server software and its components, the IDS instance does not notice directly if the NFS server crashes. Therefore, the monitoring action of Heartbeat will determine that the IDS resource works properly even if the connection to the data on the NFS share is long lost. As debugging and resolving this issue is very time consuming, it is not covered by this thesis and an according test case was not specified.

- If in a three-node cluster more than one node fails at the same time, the cluster resources are stopped and not failed over as the cluster does not have quorum anymore. This could be resolved by setting up a quorum server [LHA25] or using a different quorum plugin [LHA26]. These cases are not covered by this thesis though.

- The package installer called “yum” shipped on the RHEL5 DVD has a bug that prevents any user to use the mounted RHEL5 DVD in order to install packages. Due to IBM’s internal firewall rules the host system does not have any connection to the Internet. Therefore, the guest systems can not obtain packages from the Internet and a package installation via CD or DVD is mandatory. A workaround is to search the required packages on the DVD and install them manually via the “rpm” tool. This issue is a known bug [RedHatbug#212180].

- As the virtual test cluster is supposed to be inaccessible to the IBM intranet, the networking type “internal networking” [VBox04, chapter 6.6] is chosen in the VirtualBox VM configuration. This means that the host cannot reach the VMs via a network connection or at least a solution was not found in an appropriate time. Therefore the VirtualBox Guest Additions [VBox04, chapter 4] are installed on the guests. This enables to setup shared folders between guests and the host system, besides slightly improving the performance of the VMs.

- In the initial validation environment it was planed to use the VirtualBox GUI, but this turned out to be too unstable due to bugs causing the VirtualBox GUI to crash regularly. Therefore, the above described setup using VBoxVRDP and the “rdesktop” tool is used instead.

- Cloning a complete VM is not implemented in VirtualBox yet. A workaround is to clone the virtual hard disk image of a VM, called Virtual Disk Image (VDI) [VBox04, chapter 5.2] in VirtualBox, create a new VM and define the cloned VDI as its hard drive.

- Cloning a VDI on which SLES10 is installed and assigning the cloned VDI to a new VM leads to a new MAC address for the VM’s internal network card. This causes SLES10 to delay the boot process a huge amount of time while a configuration for the “new” network card is searched and never found. This lead to the need of processing the SLES10 for four times in order to have four SLES10 VMs. This costs a lot of time, but less than waiting for the mentioned delayed boot process to finish though.

- After updating to Heartbeat version 2.1.1 some of the command line tools did not work properly anymore. After filing a bug report [LHAbug#1661] and discussing the issue on the Linux-HA dev mailing list [LHAdevlist01] a workaround was a simple symlink. Since release 2.1.2 of Heartbeat, this bug is fixed.

- An undocumented change from Heartbeat version 2.0.8 to 2.1.1 in the IPaddr and IPaddr2 resource agents caused virtual IP address resources to not function anymore. The problem was that since Heartbeat 2.1.1 these resource agents require an additional parameter named “mac”. Setting it to “auto” resolves the issue. This bug was reported [LHAbug#1630] and is fixed since Heartbeat version 2.1.2.

- When accessing the host system via SSH with X forwarding from within a Microsoft Windows system on the IBM ThinkPad, an X server Windows port such as Cygwin/X [CygX01] is needed. Unfortunately, Cygwin/X randomly freezes the complete Windows system and a manual hardware reset of the IBM ThinkPad is necessary in order to reboot the system. A workaround is to switch to boot and use a Linux system whenever possible.

- RHEL5 comes with a kernel that has the kernel internal timer set to 1000 Hz. In contrary, on RHEL4 this timer was set to 100 Hz which is very common. The kernel timer running at 1000 Hz on RHEL5 means that the VMs process produces about ten times more system load on the host system [VBoxforums01]. This is unacceptable as it slows down all VMs and the host itself and makes working properly almost impossible. The solution is either to recompile the kernel on RHEL5 with the kernel timer set to 100 Hz or to use precompiled kernels that provide this already, such as the kernels optimized to be used as guest systems in virtualization environments provided by members of the CentOS project [CentOSkernels01].

- As mentioned above, the networking type “internal networking” is used in the VirtualBox configuration for the VMs. This causes the host system to randomly freeze while resetting, shutting down or starting one of the VMs. Unfortunately, a workaround does not exist at the moment (state of August 15th 2007). This bug is known and reported though [VBoxbug#521].

Part III – Results and Outlook

"Even a mistake may turn out to be the one thing necessary to a worthwhile achievement." – Henry Ford [Q4]

"Computers don't make errors-What they do they do on purpose." – Murphy’s Computer Laws [Q2]

"Failure is simply the opportunity to begin again, this time more intelligently." – Henry Ford [Q4]

"You will always discover errors in your work after you have printed/submitted it." – Murphy’s Computer Laws [Q2]

8. Project Results

After researching for a definition of clusters in general and explaining what High-Availability (HA) refers to, the major components involved in the final implementation are analyzed. These are in particular: IBM Informix Dynamic Server (IDS) and the High-Availability solutions that are already available for it, a detailed research, analysis and decision process on HA cluster software products for Linux, the Open Source data replication software called Distributed Replicated Block Device (DRBD) and the HA cluster software product for Linux chosen as the main component of the development process: Linux-HA.

An in-depth view on the development and validation processes is given. The end product, the IDS resource agent (RA), and the according initial specifications are described in detail as well as issues and decisions that arose during the two processes. Furthermore, the test cases (TCs), defined for the validation process, are presented and the IDS Transaction Validation Script (ITVS) especially written for the eighth test case (TC08) is analyzed in detail. A separate installation guide for the validation environment (without virtualization) is written as well.

In conclusion, the project of this thesis was an overall success. All specifications defined in the non-functional requirements specification (NFRS), functional requirements specification (FRS) and design specification (DS) are implemented and all goals are met. The IDS RA successfully passes all test cases.

In result, the IDS RA is committed to the official Linux-HA development repository [LHAdev01] and will therefore probably be a part of the upcoming official Heartbeat release. Unfortunately, a schedule for the next Heartbeat release is not defined yet. However, there already exist unofficial packages in which the IDS RA is included [LHAmlist01].

The IDS RA expands the High-Availability portfolio of IDS well and is a good complement for IDS customers that do not want to or cannot afford proprietary cluster software, but need a satisfying HA cluster solution for IDS on Linux.

9. Project Outlook

The next most reasonable step is to officially announce the IDS RA on public platforms such as the Linux-HA mailing lists [LHA24], IBM developerWorks [IBM12], the website of the International Informix User Group (IIUG) [IIUG01] and the portal called The Informix Zone [Ixzone01].

It would also be desirable to set up an entire HA cluster in a real customer scenario with further and more extensive tests than already done in the validation process. If successful customer scenarios are documented, they would be quite helpful in further promoting the IDS RA, Linux-HA and even IDS itself.

The NFRS required the solution to work on SLES10 and RHEL5, however installing and validating it on other Linux distributions, such as Ubuntu Linux, Debian GNU/Linux, Gentoo, Fedora Core, FreeBSD, OpenBSD or Sun Solaris, would be a great enrichment for the popularity of the IDS RA.

Besides the installation guide created during the validation process, screencasts and podcasts on the IDS RA in general and how to set it up and configure it would be nice to have as well.

In the technical area, it could occur that a customer wants to have an active-active scenario with the IDS RA, meaning that the IDS RA runs on several nodes at the same time while they have synchronous access to the hard disks holding the databases. IDS itself offers the feature of having several IDS instances on separate machines sharing the same storage device. The IDS RA would have to be extended in order to support this feature and make it therefore manageable as a cluster resource in Heartbeat. Another idea is to implement support for the IDS built-in features called High Availability Data Replication (HDR) and Enterprise Replication (ER).

A big step has been achieved during this thesis which serves as a good basis for further steps in the area of marketing, but also in the technical area. Or in other words: “There’s always something better just around the corner”, Author unknown.

Part IV – Appendix

"If there is any one secret of success, ... it lies in the ability in the other person's point of view and see things from that person's angle as well as from your own." – Henry Ford [Q4]

"It's not a bug, it's an undocumented feature." – Murphy’s Computer Laws [Q2]

"When everything seems to be going against you, remember that the airplane takes off against the wind, not with it." – Henry Ford [Q4]

A. Project Specifications

A.1. Non-Functional Requirements Specification (NFRS)

The non-functional requirements specification (NFRS) of the thesis “High-Availability Cluster Support for IBM Informix Dynamic Server on Linux” describes the general points important to the project. It does not define technical requirements that describe how the desired IDS resource agent for Linux-HA should work in detail, for this purpose the functional requirements specification (FRS) and the design specification (DS) exist.

A list of the non-functional requirements follows:

- Solution should depend on components as cheap as possible, Open Source products as far as possible
- Implementation including documentation is due to August 27th 2007
- Used components must be commercially usable, i.e. GPL
- Documentation of the solution implementation has to be written
- An understandable documentation on how to set up the test cluster system has to be written, screencasts are desirable though optional
- The solution must be implemented, run and tested on the provided hardware (see according chapter about the test cluster system in the documentation)
- Target operating systems the solution must be run and tested on are at least Suse Linux Enterprise Server 10 (SLES 10) and Red Hat Enterprise Linux 5 (RHEL 5), Debian GNU/Linux or Ubuntu Linux are optional
- The solution should be presented in a final presentation
- The solution has to pass the test cases which are derived from the initial use cases of the functional requirements and defined during the validation process
- The solution has to be prepared to be published as Open Source software, publishing it within the time schedule is optional

- The solution should be run and tested on a two-node and three-node cluster. The resource agent must run on one node at a time and a failover to any of the other nodes should work without errors. The resource agent does not need to be able to run in cloned or master/slave mode.

- Source code should be well-commented in order to make further development easier

- The solution should run in Heartbeat with CRM disabled and enabled, meaning with configuration syntax of Heartbeat V1 and Heartbeat V2

- The solution should run under Heartbeat version 2.0.8 and later

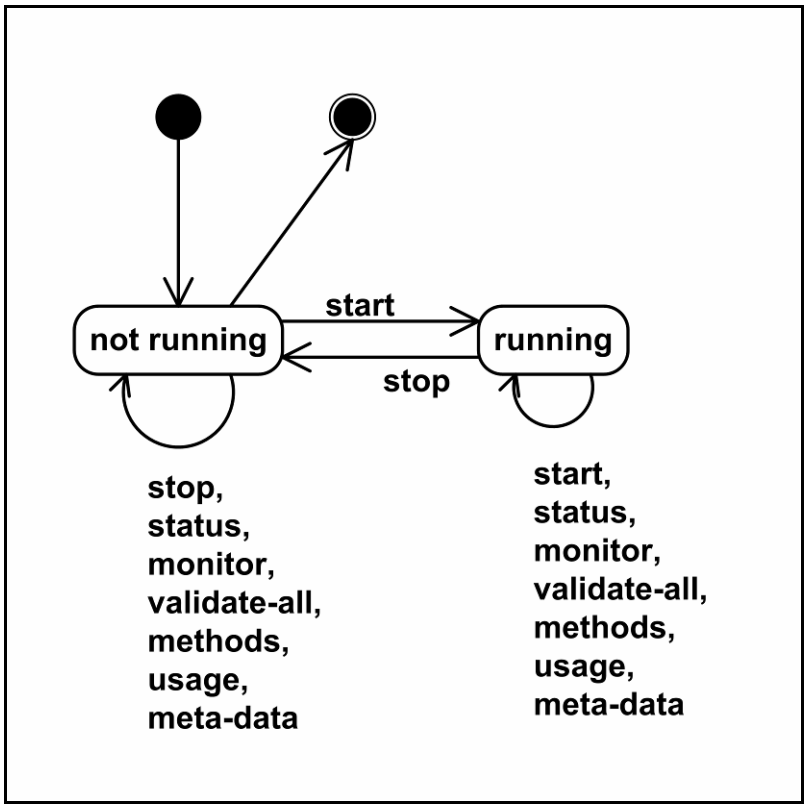
A.2. Functional Requirements Specification (FRS)

The functional requirements specification (FRS) gives a general overview on how the desired IDS resource agent for Linux-HA aka Heartbeat should work in detail. Deeper technical detail is then described in the design specification (DS). The FRS consists of the following sections:

- State transition diagram
- State transition table
- Use cases diagram
- Use case descriptions

Each section, except the use case descriptions, gives a short explanation of its purpose and how to interpret the presented graph or table in the respective section. Hereby a section always begins on a new page. A general explanation of the purpose of the use cases is given in the section of the use cases diagram.

State Transition Diagram



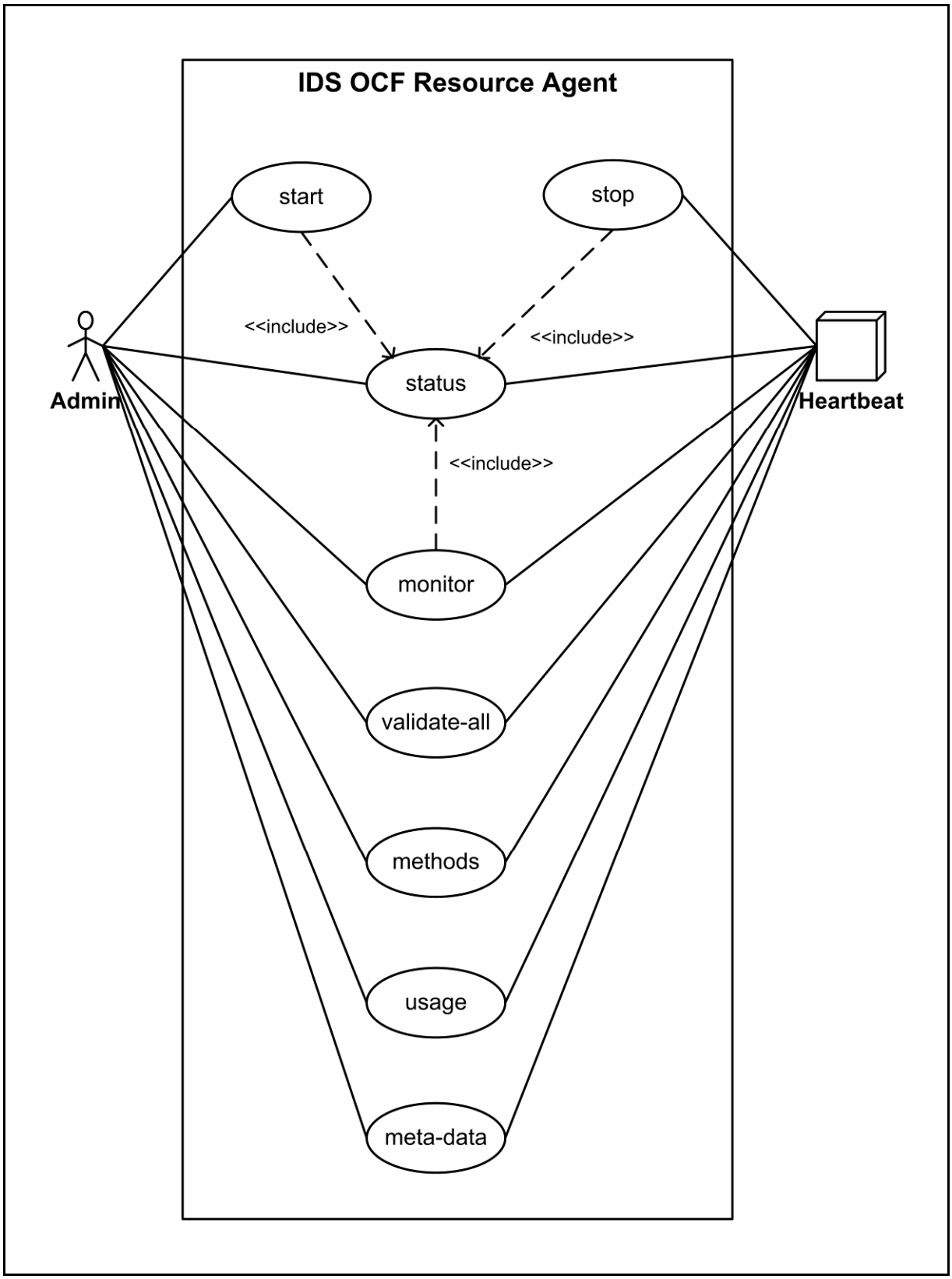
The state transition diagram represents the states the resource agent can be in when running as resource within a HA cluster. The resource thereby can be either running or not running. The initial and ending state of the resource is always not running. The state of the resource only changes from not running to running when the resource agent is invoked with the start method and vice versa. Calling the resource with any other method should not influence its status. Significant here is the fact that calling the resource in state running with the start method leaves the state of the resource untouched, the same holds for invoking the stop command in state not running.

State Transition Table

Nr. of Rule	Pre State	Command	Post State
01	not running	stop	not running
02	not running	status	not running
03	not running	monitor	not running
04	not running	validate-all	not running
05	not running	methods	not running
06	not running	usage	not running
07	not running	meta-data	not running
08	not running	start	running
09	running	start	running
10	running	status	running
11	running	monitor	running
12	running	validate-all	running
13	running	methods	running
14	running	usage	running
15	running	meta-data	running
16	running	stop	not running

The state transition table is a tabular representation of the state transition diagram above. An explanation of the different states and how they are affected by invoking the resource agent with a command is already given in the state transition graph's description above.

Use Cases Diagram



In comparison to the state transition graph and table, the use cases diagram does not represent how Heartbeat sees the resource agent as an integrated resource, but how it can invoke the resource script itself. The script offers eight methods, called use cases here, which are the following:

- **start**
  The start method starts the resource, respectively an IDS instance. Before doing that it checks the current status and only attempts to start the resource if it is not already running.

- **stop**
  The stop method stops the resource, respectively the running IDS instance. Before doing that it checks the current status and only attempts to stop the resource if it is running.

- **status**
  This method checks the current status of the resource, meaning whether the managed IDS instance is running or not running and returns the result.

- **monitor**
  This method invokes the status method and depending on the status' result tries to execute a test SQL query to the IDS database server instance. The result is then returned.

- **validate-all**
  The validate-all method verifies the passed configuration parameters and returns an according exit status code indicating a valid or invalid configuration of the resource agent.

- **methods**
  This method simply returns a list of the methods provided by the script.

- **usage**
  This method returns a short general explanation of the syntax in which the resource agent expects to be called. This also shows in which order the configuration parameters are expected.

- **meta-data**
  This method returns a description of the script and explanations about the expected configuration parameters in XML format.

In order to avoid redundant source code later on, it is pointed out that the use cases start, stop and monitor make use of, that is include, the status use case. As the resource agent script can be either called manually by an administrator (or any other

person that has the appropriate permissions) or by the Heartbeat process, the use cases diagram above shows two actors: Admin and Heartbeat. Once the desired resource agent script is implemented and validated only Heartbeat will be calling the resource agent in normal circumstances though. The rest of this document gives a more detailed overview on the use cases described above.

Use Case 01 – start

Name	Use Case 01 - start
Description	Starts an instance of IDS
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “start” command
Incoming Information	Environment variables: INFORMIXDIR, INFORMIXSERVER, ONCONFIG
Outgoing Information	Exit status code indicating success or failure for starting the IDS instance
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “start” 2) The IDS resource agent script verifies the three necessary environment variables (INFORMIXDIR, INFORMIXSERVER and ONCONFIG). If the variables are valid, the script continues with step 3) 3) The current status of the IDS instance is determined → *Use case 03 - status*. When the called resource is not running yet, the script continues with step 4) 4) The script tries to start an instance of IDS 5) The status of the IDS instance is determined again → *Use case 03 – status*. If the IDS instance is now running, the script terminates with an exit status code

	indicating success
Alternate Flows	2a) If the variables are not valid, the script will write an according entry into the logfiles and terminate with an error exit status code 3a) When the IDS resource is already running, nothing is changed and the script will terminate with an exit status code indicating success 5a) If the IDS instance is not running after step 4), the script terminates with an error exit status code

Use Case 02 – stop

Name	Use Case 02 - stop
Description	Stops a running instance of IDS
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “stop” command
Incoming Information	Environment variables: INFORMIXDIR, INFORMIXSERVER, ONCONFIG
Outgoing Information	Exit status code indicating success or failure for stopping the IDS instance
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “stop” 2) The IDS resource agent script verifies the three necessary environment variables (INFORMIXDIR, INFORMIXSERVER and ONCONFIG). If the variables are valid, the script continues with step 3) 3) The current status of the IDS instance is determined → *Use case 03 – status*. When the called resource is

	running, the script continues with step 4) 4) The script tries to stop the instance of IDS 5) The status of the IDS instance is determined again → *Use case 03* – status. If the status of the IDS instance indicates now that it's not running anymore, the script terminates with an exit status code indicating success
Alternate Flows	2a) If the variables are not valid, the script will write an according entry into the logfiles and terminate with an error exit status code 3a) When the IDS resource is not running, nothing is changed and the script will terminate with an exit status code indicating success 5a) If the IDS instance is still running after step 4), the script terminates with an error exit status code

Use Case 03 – status

Name	Use Case 03 - status
Description	Determines and returns the status of an IDS instance
Actors	Admin, Heartbeat or IDS resource agent script (via include)
Trigger	The IDS resource agent called with "status" command
Incoming Information	Environment variables: INFORMIXDIR, INFORMIXSERVER, ONCONFIG
Outgoing Information	Exit status code indicating the current status of the IDS instance
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin, Heartbeat or the IDS resource agent script itself (called with a different method than status) calls the IDS resource agent with method "status"

	2) The IDS resource agent script verifies the three necessary environment variables (INFORMIXDIR, INFORMIXSERVER and ONCONFIG). If the variables are valid, the script continues with step 3) 3) The status of the IDS instance is determined and the script terminates with an appropriate exit status code
Alternate Flows	2a) If the variables are not valid, the script will write an according entry into the logfiles and terminate with an error exit status code

Use Case 04 – monitor

Name	Use Case 04 - monitor
Description	Monitors a running instance of IDS
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “monitor” command
Incoming Information	Environment variables: INFORMIXDIR, INFORMIXSERVER, ONCONFIG
Outgoing Information	Exit status code indicating success or failure for monitoring the IDS instance
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “monitor” 2) The IDS resource agent script verifies the three necessary environment variables (INFORMIXDIR, INFORMIXSERVER and ONCONFIG). If the variables are valid, the script continues with step 3) 3) The current status of the IDS instance is determined → *Use case 03 – status*. When the called resource is

	running, the script continues with step 4) 4) An example SQL query is sent to the IDS instance 5) If the query in step 4) returns an exit status code of success, the script terminates with the same success exit status code
Alternate Flows	2a) If the variables are not valid, the script will write an according entry into the logfiles and terminate with an error exit status code 3a) If the IDS resource is not running, nothing is changed and the script will terminate with an exit status code indicating that the resource is not running 5a) If the query in step 4) is not successful, the script terminates with a exit status code indicating failure

Use Case 05 – validate-all

Name	Use Case 05 – validate-all
Description	Validates the parameters (see field “Incoming Information”) passed to the IDS resource agent
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “validate-all” command
Incoming Information	Environment variables: INFORMIXDIR, INFORMIXSERVER, ONCONFIG
Outgoing Information	Exit status code indicating if the parameters (see field “Incoming Information”) passed to the IDS resource agent are valid or not
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “validate-all”

	2) The IDS resource agent script verifies the three necessary environment variables (INFORMIXDIR, INFORMIXSERVER and ONCONFIG). If the variables are valid, the script terminates returning an exit status code of success
Alternate Flows	2a) If the variables are not valid, the script will write an according entry into the logfiles and terminate with an error exit status code

Use Case 06 – methods

Name	Use Case 06 – methods
Description	Returns a list of the methods the IDS resource agent offers
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “methods” command
Incoming Information	-
Outgoing Information	A list of methods the IDS resource agent offers
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “methods” 2) The IDS resource agent script returns a list of offered methods and terminates
Alternate Flows	-

Use Case 07 – usage

Name	Use Case 07 – usage
Description	Returns an explanation on how to use the IDS resource agent, a list of the supported methods and their meanings
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “usage” command
Incoming Information	-
Outgoing Information	Usage explanation for the IDS resource agent
Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “usage” 2) The IDS resource agent script returns an explanation on how to use the script and terminates
Alternate Flows	-

Use Case 08 – meta-data

Name	Use Case 08 – meta-data
Description	Returns the XML meta-data of the IDS resource agent
Actors	Admin or Heartbeat
Trigger	The IDS resource agent called with “meta-data” command
Incoming Information	-
Outgoing Information	XML meta-data of the IDS resource agent script

Precondition	IDS installed and configured correctly, IDS Linux-HA resource configured correctly
Basic Flow	1) The Admin or Heartbeat call the IDS resource agent with method “meta-data” 2) The IDS resource agent script returns the XML meta-data and terminates
Alternate Flows	-

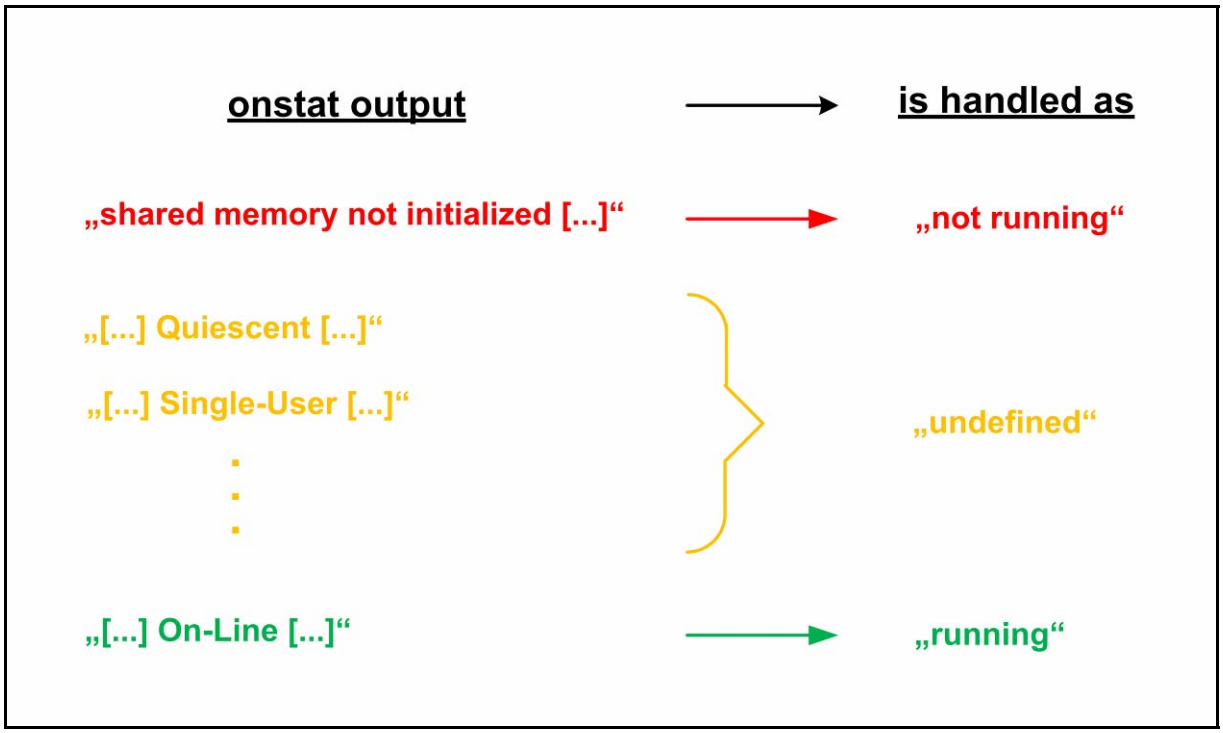
A.3. Design Specification (DS)

The design specification (DS) gives a more in-depth view on the technical aspects of the desired IDS resource agent than the functional requirements specification (FRS). In the DS, concrete implementation decisions on the behavior of the resource script are made and specified. The DS consists of the following sections:

- Sate definitions
- State transition diagram
- State transition table
- Flow charts

Each section gives a short explanation of its purpose and how to interpret the presented graph or table in the respective section. Each section begins on a new page. As the flow charts of the methods status, usage, methods and meta-data are quite simple, they are left out here. Instead the DS concentrates on the flow charts of the methods start, stop, validate-all and monitor.

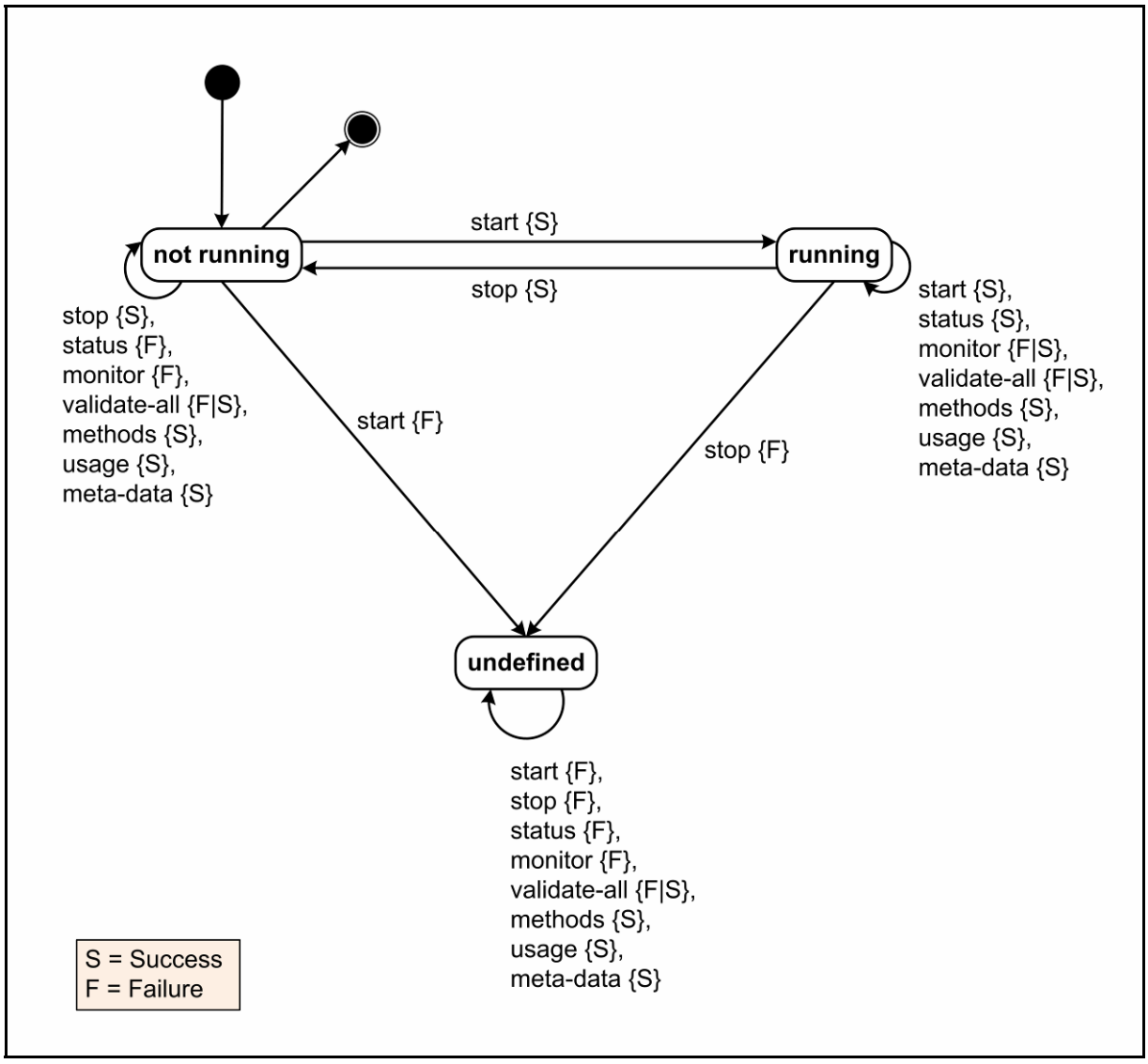
State Definitions



The graph above explains how the resource agent will interpret the different states an IDS instance can be in. The status of an IDS instance is thereby determined by the onstat command. The resource agent will identify three different state groups: not running, undefined and running. If IDS is not running, onstat will return a message containing the text “shared memory not initialized […]” for which the resource agent defines the resource in Heartbeat as currently not running. If onstat returns a message containing the text “[…] On-Line […]”, the IDS instance is online and therefore the resource in Heartbeat is considered to be running too.

In any other case, the resource agent will define the state of the resource as undefined and react accordingly by eventually taking measures and returning an error by all means.

State Transition Diagram



The state transition diagram shows the possible changes in state and the returned exit status code by the script indicating success (S) or failure (F). As in the state transition diagram of the FRS, the initial and ending state of an IDS resource in Heartbeat is not running. Only the method start can cause the not running state to change. If the start procedure is successful, the new state will be running and an exit status code indicating success is returned. A failure during the start procedure leads to the new state being undefined and returning an exit status code indicating failure. This is analogous when invoking the resource agent with the stop method when the current state is running. Of course, starting an already started resource does not

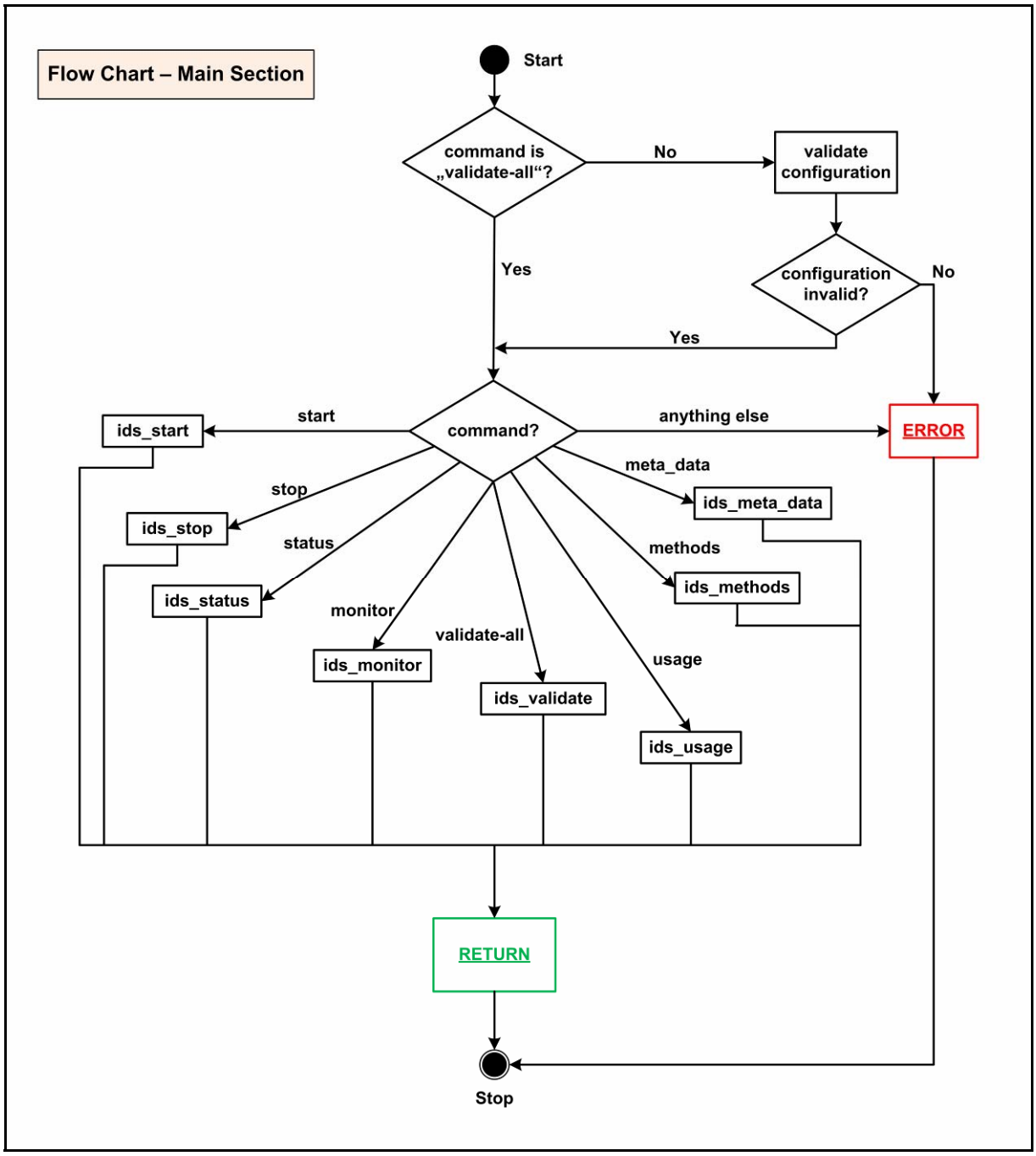
change the state and always returns success, same holds for stopping a not running resource. Important to point out here, is the fact that invoking the script with the start, stop, status or monitor method while being in state undefined will not effect the state but will always return an exit status code indicating error. This is implemented in this way in order to have the administrator analyze the issue as it is obvious that the IDS instance is not behaving as expected and probably intended. The ending state of the diagram is defined to be in state not running as Heartbeat is configured by default to drop resources it cannot start which leads to the resource being marked as not running in the end. This fact is not explicitly pointed out in the diagram though.

State Transition Table

Nr. of Rule	Pre State	Command	Exit Code	Post State
01	not running	stop	success	not running
02	not running	status	failure	not running
03	not running	monitor	failure	not running
04	not running	validate-all	failure or success	not running
05	not running	methods	success	not running
06	not running	usage	success	not running
07	not running	meta-data	success	not running
08	not running	start	failure	undefined
09	not running	start	success	running
10	running	stop	success	not running
11	running	stop	failure	undefined
12	running	start	success	running
13	running	status	success	running
14	running	monitor	failure or success	running
15	running	validate-all	failure or success	running
16	running	methods	success	running
17	running	usage	success	running
18	running	meta-data	success	running
19	undefined	start	failure	undefined
20	undefined	stop	failure	undefined
21	undefined	status	failure	undefined
22	undefined	monitor	failure	undefined
23	undefined	validate-all	failure or success	undefined
24	undefined	methods	success	undefined
25	undefined	usage	success	undefined
26	undefined	meta-data	success	undefined

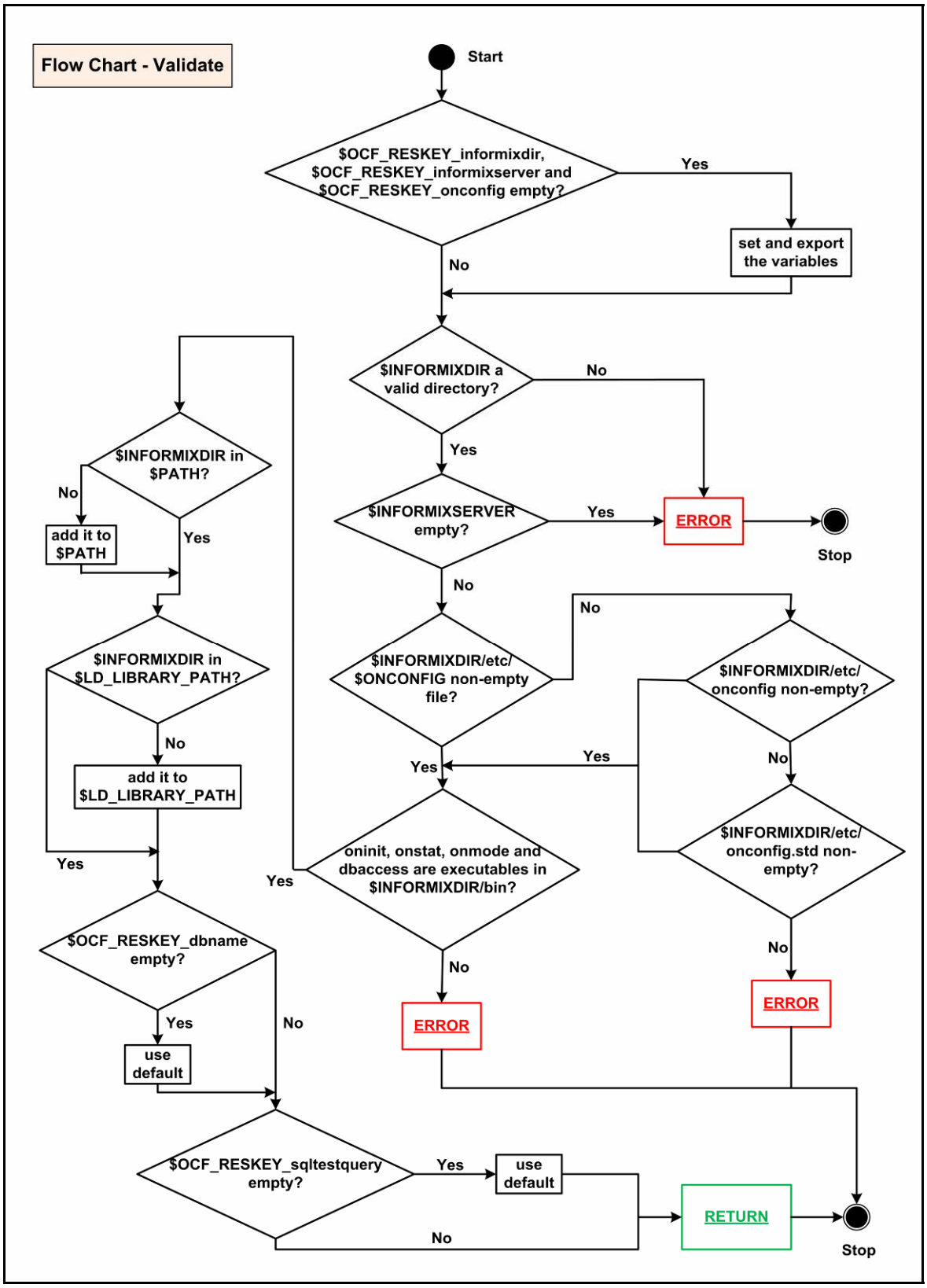
The state transition table is a tabular representation of the state transition diagram. A detailed explanation was already given above.

Flow Chart – Main Section



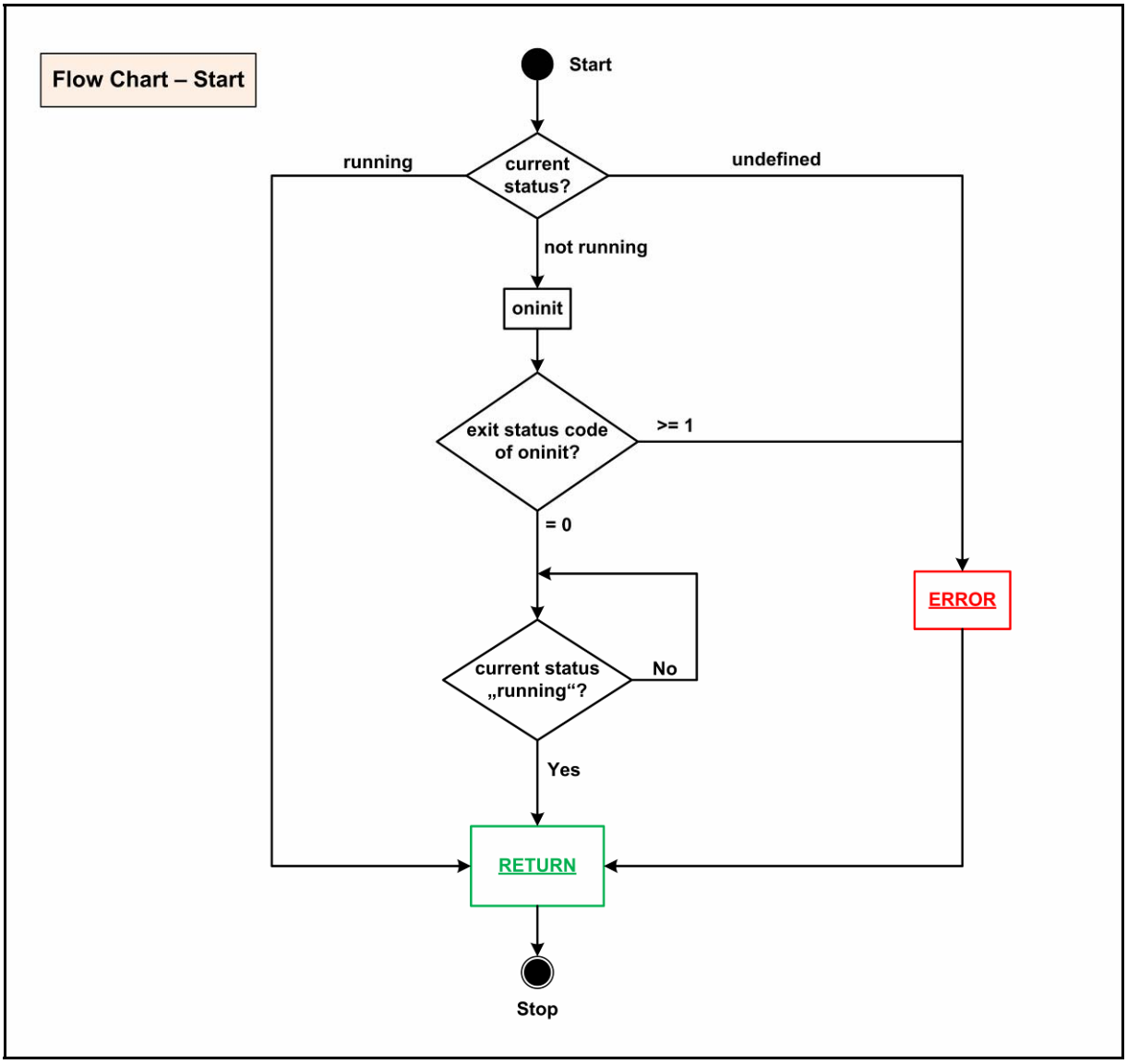
This flow chart represents how the main section of the script looks like. Before executing any given method the configuration is validated by calling the method validate-all. In case of success the passed method is executed and its exit status code is returned. If the given method does not match any of the eight defined methods, an exit status code indicating failure is returned.

Flow Chart – Validate



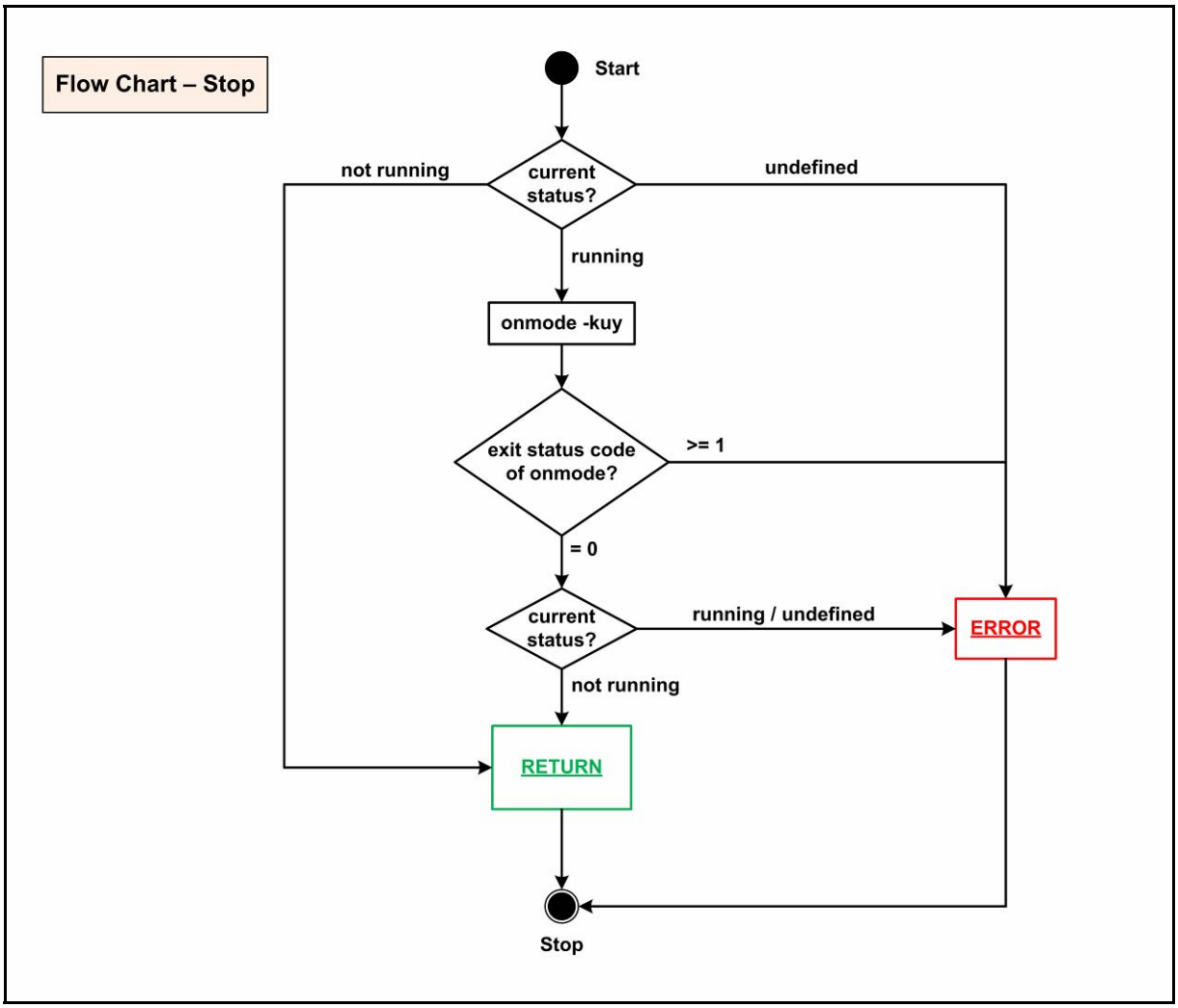
The validate-all method is the most complex one of all eight defined methods and therefore its flow chart is the largest and most complex. The method checks if the shell environment variables that IDS needs in order to run are set correctly. In some cases the method thereby tries to switch to default values if possible. If this succeeds the method returns an exit status code indicating success which means the configuration of the IDS resource is considered as valid. If checking or setting of one of the required variables fails, the method terminates with an exit status code indicating failure, which is interpreted as an invalid resource agent configuration.

Flow Chart – Start



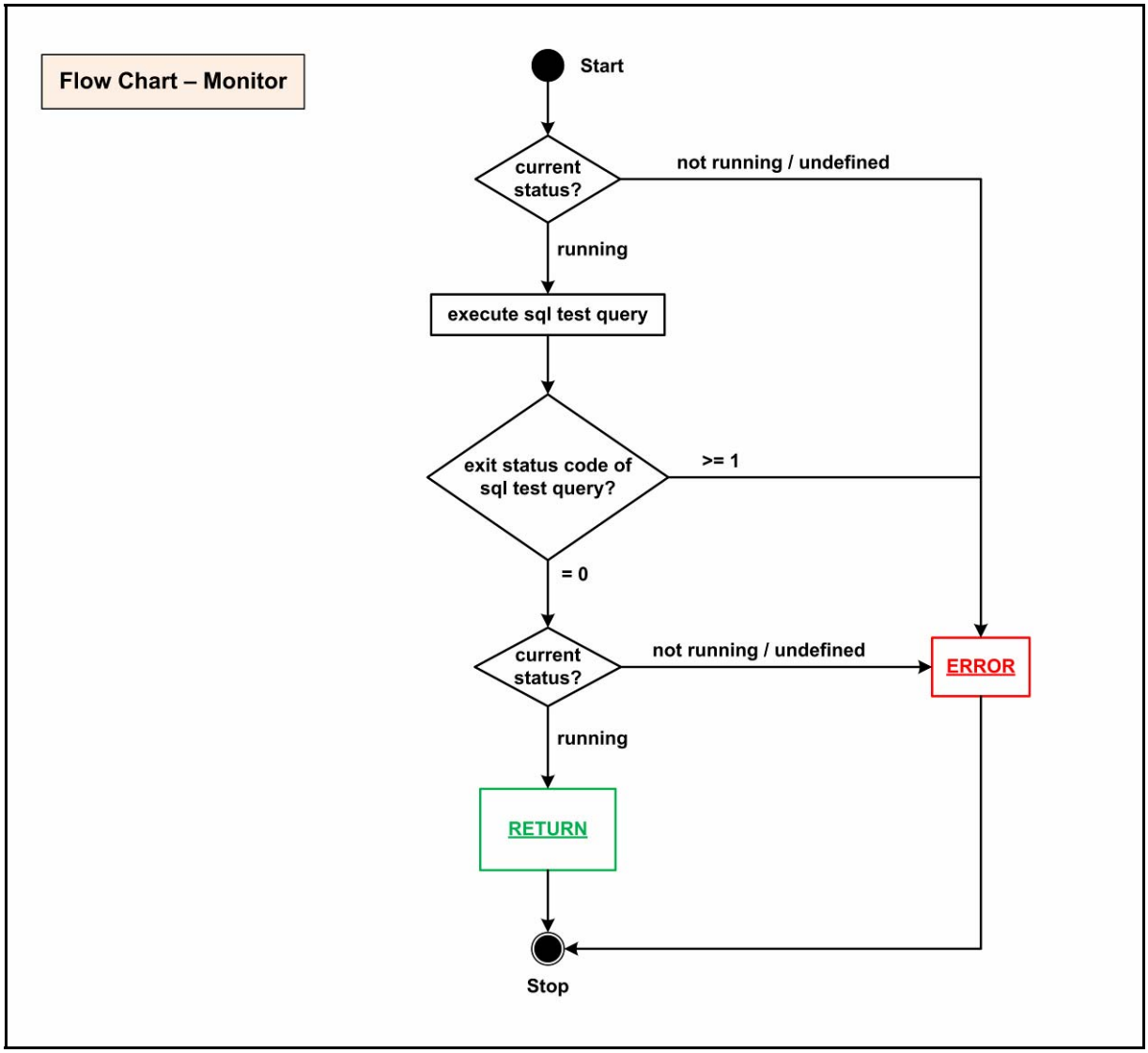
The start method checks the current status by calling the status method in order to determine how to proceed further. This is done in order to cover the case where the resource agent is invoked with the method start while the resource is already running. In this case running the IDS start procedure can be avoided and an exit status code indicating success is returned directly. An undefined state leads to directly terminating the method and returning an exit status code indicating failure. If the status method returns an exit status code indicating that the resource is not running, the IDS start procedure is run via the oninit command. In the case that the oninit command returns an exit status code indicating failure, the start method also return a failure. In the other case the start method jumps into an endless loop that checks the current status of the resource until it is running. This can be implemented in this way, because Heartbeat has certain timeouts for starting a resource and will terminate the script after the configured timeout is reached. If the endless loop terminates because of the new status of the resource being running, the start method exits with an exit status code indicating success.

Flow Chart – Stop



The start method's duty is it to stop the currently running IDS resource. Of course, if the resource is not running the method terminates directly returning an exit status code indicating success. An undefined state leads to directly terminating the method and returning an exit status code indicating failure. If the resource is considered to be running, the IDS stop procedure is run via the onmode command and the according parameters. If the onmode command terminates with an error, the method terminates with an error as well. Otherwise the new status of the resource is checked. If the resource status is then running or undefined the method terminates with an error, else it terminates with an exit status code indicating that the resource is successfully stopped.

Flow Chart – Monitor



The monitor method can be considered as a kind of advanced status method. The first thing it does is to determine the current status of the resource via the status method. If the resource is not running or in an undefined state the method terminates with an error. Otherwise it executes a SQL test query on the IDS database server in order to check if it is fully functional besides being online. If the SQL test query does not succeed the method terminates with an error. If the SQL test query succeeds the status method is invoked again. If the status of the resource is still running the monitor method terminates with an exit status code indicating success, otherwise it will indicate an error.

A.4. Test Cases (TCs)

In order to validate if the functional requirements specification (FRS) and the design specification (DS) are implemented correctly, the test cases (TCs) in this document were defined as a preparation for the validation phase. Besides representing common scenarios in high-availability (HA) cluster, they also assure that the IDS resource agent (RA) is fully compliant with the OCF standard. As required by the non-functional requirements specification (NFRS), the TCs are performed on Suse Linux Enterprise Server 10 (SLES10) and Red Hat Enterprise Linux 5 (RHEL5). A test case consists of:

- **Test Case ID** – in order to uniquely identify each test case
- **Description** – the current situation and actions being taken
- **Expected result** – the expected result of the test case
- **Output on SLES10** – console and log file output on SLES10
- **Output on RHEL5** – console and log file output on RHEL5
- **Results on SLES10** – how and if the cluster behaved as expected
- **Results on RHEL5** – how and if the cluster behaved as expected

Thereby the outputs from the console and log file outputs are printed in the Courier New font. In addition, the console outputs are highlighted with a gray background in order to make them distinguishable from the log file outputs. Furthermore, each test case begins on a new page.

For test case TC08 a special test script, named IDS Transaction Validation Script (ITVS), was written as a shell script. The source code of this shell script is attached on the CD-ROM in Appendix D. A detailed functional description of the script is given in chapter 7.4 within the thesis itself.

Test Case 01 (TC01)

Test Case ID: TC01

Description:

Pass the IDS resource agent to the ocf-tester script that is shipped with Heartbeat in order to verify functionality and compliance with the OCF standard. The shared storage on which the IDS database resides on and the virtual cluster IP have to be assigned manually to the node running this test on.

Expected Results:

No errors should be reported by the ocf-tester script for the ids resource agent.

Output on SLES10:

```
sles10-node1:~ # export OCF_ROOT=/usr/lib/ocf && /usr/sbin/ocf-tester -v -
n ids /usr/lib/ocf/resource.d/ibm/ids
Beginning tests for usr/lib/ocf/resource.d/ibm/ids...
Testing: meta-data
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="ids">
[...]
</resource-agent>
Testing: validate-all
Checking current state
Testing: monitor
Testing: start
Testing: monitor
Testing: notify
2007/07/30_16:13:18 ERROR: mainsection: no or invalid command supplied:
notify
* Your agent does not support the notify action (optional)
Checking for demote action
2007/07/30_16:13:19 ERROR: mainsection: no or invalid command supplied:
demote
* Your agent does not support the demote action (optional)
Checking for promote action
2007/07/30_16:13:20 ERROR: mainsection: no or invalid command supplied:
promote
* Your agent does not support the promote action (optional)
* Your agent does not support master/slave (optional)
Testing: stop
Testing: monitor
Restarting resource...
Testing: monitor
Testing: starting a started resource
Testing: monitor
Stopping resource
Testing: monitor
Testing: stopping a stopped resource
Testing: monitor
Checking for migrate_to action
2007/07/30_16:13:44 ERROR: mainsection: no or invalid command supplied:
migrate_to
Checking for reload action
```

```
2007/07/30_16:13:44 ERROR: mainsection: no or invalid command supplied:
reload
* Your agent does not support the reload action (optional)
/usr/lib/ocf/resource.d/ibm/ids passed all tests
sles10-node1:~ #
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

- ✓ The IDS RA is fully OCF compliant and functional according to the Heartbeat ocf-tester script. Note that in the output the action “migrate_to” is not marked as optional. This is fixed in newer releases of Linux-Ha.

Results on RHEL5:

- ✓ The IDS RA is fully OCF compliant and functional according to the Heartbeat ocf-tester script. Note that in the output the action “migrate_to” is not marked as optional. This is fixed in newer releases of Linux-Ha.

Test Case 02 (TC02)

Test Case ID: TC02

Description:

Call the IDS RA manually with the commands start, stop and status and verify the output via the output of the onstat tool shipped with IDS. The shared storage on which the IDS database resides on and the virtual cluster IP have to be assigned manually to the node running this test on.

Expected Results:

The states the IDS RA returns match with the output of the onstat tool.

Output on SLES10:

```
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids status; echo $?
7
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids start; echo $?
0
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids status; echo $?
0
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
```

```
00:00:19 -- 28288 Kbytes
sles10-node1:~ # onmode -j
This will change mode to single user. Only DBSA/informix can connect in
this mode.
Do you wish to continue (y/n)? y
All threads which are not owned by DBSA/informix will be killed.
Do you wish to continue (y/n)? y
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- Single-User -- Up
00:00:36 -- 28288 Kbytes
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids status; echo $?
2007/07/30_16:24:12 ERROR: ids_status: IDS instance status undefined:
IBM Informix Dynamic Server Version 11.10.UB7     -- Single-User -- Up
00:00:41 -- 28288 Kbytes
1
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids stop; echo $?
2007/07/30_16:24:32 ERROR: ids_status: IDS instance status undefined:
IBM Informix Dynamic Server Version 11.10.UB7     -- Single-User -- Up
00:01:01 -- 28288 Kbytes
2007/07/30_16:24:32 ERROR: ids_stop: IDS instance in undefined state: 1
1
sles10-node1:~ # onmode -m
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
00:02:00 -- 28288 Kbytes
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids status; echo $?
0
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids stop; echo $?
0
sles10-node1:~ # /usr/lib/ocf/resource.d/ibm/ids status; echo $?
7
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ #
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected.

Results on RHEL5:

✓ As expected.

Test Case 03 (TC03)

Test Case ID: TC03
Description: Any number of nodes is online and IDS is running as a resource on one of them. A monitoring action is defined for the IDS resource. Then kill the IDS process.
Expected Results:

The monitoring action notices that the IDS resource is not running and restarts it. After that, IDS should be running again.

Output on SLES10:

```
sles10-node1:~ # onmode -kuy
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- Initialization -- Up
00:00:07 -- 28288 Kbytes
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- Fast Recovery -- Up
00:00:11 -- 28288 Kbytes
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
00:00:13 -- 28288 Kbytes
sles10-node1:~ #
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected.

Results on RHEL5:

✓ As expected.

Test Case 04 (TC04)

Test Case ID: TC04

Description:

Start IDS manually before starting Heartbeat which then tries to (re)start IDS. The shared storage on which the IDS database resides on and the virtual cluster IP have to be assigned manually to the node running this test on.

Expected Results:

Heartbeat will conclude that IDS has been successfully started.

Output on SLES10:

```
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ #
sles10-node1:~ # ifconfig eth0:1 192.168.15.253
```

```
sles10-node1:~ # mount sles10-san:/san /mnt/san/
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ # oninit
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
00:00:14 -- 28288 Kbytes
sles10-node1:~ # /etc/init.d/heartbeat start
Starting High-Availability services: done
sles10-node1:~ # crm_mon -1
============
Last updated: Mon Jul 30 17:15:44 2007
Current DC: sles10-node1 (d0870d17-a7b2-4b76-a3ac-23343f8e8f73)
3 Nodes configured.
3 Resources configured.
============
Node: sles10-node1 (d0870d17-a7b2-4b76-a3ac-23343f8e8f73): online
Node: sles10-node2 (3562a151-17d7-4fd6-8df0-f2f995c4e83c): online
Node: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995): OFFLINE
Resource Group: ids_validation_cluster
    cNFS        (heartbeat::ocf:Filesystem):    Started sles10-node1
    cIP (heartbeat::ocf:IPaddr2):       Started sles10-node1
    cIDS        (ibm::ocf:ids): Started sles10-node1
Clone Set: pingd
    pingd-child:0       (heartbeat::ocf:pingd): Started sles10-node1
    pingd-child:1       (heartbeat::ocf:pingd): Started sles10-node2
    pingd-child:2       (heartbeat::ocf:pingd): Stopped
stonith_meatware        (stonith:meatware):     Started sles10-node2
sles10-node1:~ #
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected.

Results on RHEL5:

✓ As expected.

Test Case 05 (TC05)

Test Case ID: TC05

Description:

Bring IDS manually in an undefined state (i.e. single-user mode or quiescent mode).

Expected Results:

This causes the monitoring action to fail and Heartbeat will try to stop the IDS resource which will fail as well. Then one of the other nodes will define the node IDS ran on as dead and try to shut it down before failing the resource over.

Output on SLES10:

On node sles10-node1:

```
sles10-node1:~ # onstat -
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
00:01:18 -- 28288 Kbytes
sles10-node1:~ # onmode -j
This will change mode to single user. Only DBSA/informix can connect in
this mode.
Do you wish to continue (y/n)? y
All threads which are not owned by DBSA/informix will be killed.
Do you wish to continue (y/n)? y
sles10-node1:-# crm_mon -1
============
Last updated: Mon Jul 30 17:24:25 2007
Current DC: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995)
3 Nodes configured.
3 Resources configured.
============
Node: sles10-node1 (d0870d17-a7b2-4b76-a3ac-23343f8e8f73): online
Node: sles10-node2 (3562a151-17d7-4fd6-8df0-f2f995c4e83c): online
Node: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995): online
Clone Set: pingd
    pingd-child:0       (heartbeat::ocf:pingd): Started sles10-node2
    pingd-child:1       (heartbeat::ocf:pingd): Started sles10-node3
    pingd-child:2       (heartbeat::ocf:pingd): Started sles10-node1
stonith_meatware        (stonith:meatware):     Started sles10-node3
Failed actions:
    cIDS_stop_0 (node=sles10-node1, call=20, rc=1): Error
sles10-node1:~ #
```

Excerpt from /var/log/messages on node sles10-node1:

```
Jul 30 17:20:00 sles10-node1 crmd: [32651]: info: do_lrm_rsc_op:
Performing op=cIDS_monitor_10000 key=20:7:ac2f114d-63f5-42a5-b132-
e2c6a0fb5cd3)

Jul 30 17:20:11 sles10-node1 crmd: [32651]: info: process_lrm_event: LRM
operation cIDS_monitor_10000 (call=19, rc=0) complete

Jul 30 17:21:15 sles10-node1 ids[912]: [926]: ERROR: ids_status: IDS
instance status undefined:  IBM Informix Dynamic Server Version 11.10.UB7
-- Single-User -- Up 00:01:27 -- 28288 Kbytes

Jul 30 17:21:15 sles10-node1 ids[912]: [930]: ERROR: ids_monitor: IDS
instance in undefined state: 1

Jul 30 17:21:15 sles10-node1 crmd: [32651]: ERROR: process_lrm_event: LRM
operation cIDS_monitor_10000 (call=19, rc=1) Error unknown error

Jul 30 17:21:17 sles10-node1 crmd: [32651]: info: do_lrm_rsc_op:
Performing op=cIDS_stop_0 key=2:9:ac2f114d-63f5-42a5-b132-e2c6a0fb5cd3)

Jul 30 17:21:17 sles10-node1 crmd: [32651]: info: process_lrm_event: LRM
operation cIDS_monitor_10000 (call=19, rc=-2) Cancelled

Jul 30 17:21:17 sles10-node1 ids[953]: [972]: ERROR: ids_status: IDS
instance status undefined:  IBM Informix Dynamic Server Version 11.10.UB7
-- Single-User -- Up 00:01:29 -- 28288 Kbytes

Jul 30 17:21:17 sles10-node1 ids[953]: [977]: ERROR: ids_stop: IDS
instance in undefined state: 1
```

```
Jul 30 17:21:17 sles10-node1 crmd: [32651]: ERROR: process_lrm_event: LRM
operation cIDS_stop_0 (call=20, rc=1) Error unknown error
```

Excerpt of /var/log/messages on node sles10-node3:

```
Jul 30 17:29:55 sles10-node3 stonithd: [5996]: info: client tengine [pid:
6092] want a STONITH operation RESET to node sles10-node1.

Jul 30 17:29:55 sles10-node3 tengine: [6092]: info: te_pseudo_action:
Pseudo action 30 fired and confirmed

Jul 30 17:29:55 sles10-node3 tengine: [6092]: info: te_fence_node:
Executing reboot fencing operation (34) on sles10-node1 (timeout=30000)

Jul 30 17:29:55 sles10-node3 stonithd: [5996]: info:
stonith_operate_locally::2532: sending fencing op (RESET) for sles10-node1
to device meatware (rsc_id=stonith_meatware, pid=6505)

Jul 30 17:29:55 sles10-node3 stonithd: [6505]: CRIT: OPERATOR INTERVENTION
REQUIRED to reset sles10-node1.

Jul 30 17:29:55 sles10-node3 stonithd: [6505]: CRIT: Run "meatclient -c
sles10-node1" AFTER power-cycling the machine.
```

On node sles10-node3:

```
sles10-node3:~ # meatclient -c sles10-node1
WARNING!
If node "sles10-node1" has not been manually power-cycled or disconnected
from all shared resources and networks, data on shared disks may become
corrupted and migrated services might not work as expected.
Please verify that the name or address above corresponds to the node you
just rebooted.
PROCEED? [yN] y
Meatware_client: reset confirmed.
sles10-node3:~ # crm_mon -1
============
Last updated: Mon Jul 30 17:35:25 2007
Current DC: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995)
3 Nodes configured.
3 Resources configured.
============
Node: sles10-node1 (d0870d17-a7b2-4b76-a3ac-23343f8e8f73): OFFLINE
Node: sles10-node2 (3562a151-17d7-4fd6-8df0-f2f995c4e83c): online
Node: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995): online
Resource Group: ids_validation_cluster
    cNFS        (heartbeat::ocf:Filesystem):    Started sles10-node2
    cIP (heartbeat::ocf:IPaddr2):       Started sles10-node2
    cIDS        (ibm::ocf:ids): Started sles10-node2
Clone Set: pingd
    pingd-child:0       (heartbeat::ocf:pingd): Started sles10-node2
    pingd-child:1       (heartbeat::ocf:pingd): Started sles10-node3
    pingd-child:2       (heartbeat::ocf:pingd): Stopped
stonith_meatware        (stonith:meatware):     Started sles10-node3
sles10-node3:~ #
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected.

Results on RHEL5:

✓ As expected.

Test Case 06 (TC06)

Test Case ID: TC06

Description:

Three nodes are online and the one running the IDS resource fails by disconnecting it from the network (via unplugging the network cable or shutting down the network interface). This same test is rerun by also failing the node by simply rebooting it and manually killing the heartbeat processes.

Expected Results:

In all above described variants the result is the same: One of the two remaining nodes shuts down the failed node via STONITH and the resources are failed over.

Output on SLES10:

On the node sles10-node1:

```
ifconfig eth0 down
```

Excerpt of /var/log/messages on the node sles10-node3:

```
Jul 30 17:49:55 sles10-node3 stonithd: [5996]: info: client tengine [pid:
6092] want a STONITH operation RESET to node sles10-node1.

Jul 30 17:49:55 sles10-node3 tengine: [6092]: info: te_pseudo_action:
Pseudo action 30 fired and confirmed

Jul 30 17:49:55 sles10-node3 tengine: [6092]: info: te_fence_node:
Executing reboot fencing operation (34) on sles10-node1 (timeout=30000)

Jul 30 17:49:55 sles10-node3 stonithd: [5996]: info:
stonith_operate_locally::2532: sending fencing op (RESET) for sles10-node1
to device meatware (rsc_id=stonith_meatware, pid=6505)

Jul 30 17:49:55 sles10-node3 stonithd: [6505]: CRIT: OPERATOR INTERVENTION
REQUIRED to reset sles10-node1.

Jul 30 17:49:55 sles10-node3 stonithd: [6505]: CRIT: Run "meatclient -c
sles10-node1" AFTER power-cycling the machine.
```

On the node sles10-node3:

```
sles10-node3:~ # meatclient -c sles10-node1
WARNING!
If node "sles10-node1" has not been manually power-cycled or disconnected
```

```
from all shared resources and networks, data on shared disks may become
corrupted and migrated services might not work as expected.
Please verify that the name or address above corresponds to the node you
just rebooted.
PROCEED? [yN] y
Meatware_client: reset confirmed.
sles10-node3:~ # crm_mon -1
============
Last updated: Mon Jul 30 17:35:25 2007
Current DC: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995)
3 Nodes configured.
3 Resources configured.
============
Node: sles10-node1 (d0870d17-a7b2-4b76-a3ac-23343f8e8f73): OFFLINE
Node: sles10-node2 (3562a151-17d7-4fd6-8df0-f2f995c4e83c): online
Node: sles10-node3 (77bf4db1-4959-4ab1-82fc-96afea972995): online
Resource Group: ids_validation_cluster
    cNFS        (heartbeat::ocf:Filesystem):    Started sles10-node2
    cIP (heartbeat::ocf:IPaddr2):       Started sles10-node2
    cIDS        (ibm::ocf:ids): Started sles10-node2
Clone Set: pingd
    pingd-child:0       (heartbeat::ocf:pingd): Started sles10-node2
    pingd-child:1       (heartbeat::ocf:pingd): Started sles10-node3
    pingd-child:2       (heartbeat::ocf:pingd): Stopped
stonith_meatware        (stonith:meatware):     Started sles10-node3
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ All three variants of the test terminate as expected.

Results on RHEL5:

✓ All three variants of the test terminate as expected.

Test Case 07 (TC07)

Test Case ID: TC07

Description:

Remove IDS resource from active cluster.

Expected Results:

IDS is supposed to be automatically stopped before removing it from the cluster resource configuration.

Output on SLES10:

On node sles10-node1:

```
sles10-node1:~ # onstat -
```

```
IBM Informix Dynamic Server Version 11.10.UB7     -- On-Line -- Up
00:01:37 -- 28288 Kbytes
sles10-node1:~ # cibadmin -D -o resources -X '<primitive id="cIDS" />'
sles10-node1:~ # onstat -
shared memory not initialized for INFORMIXSERVER 'ids1'
sles10-node1:~ #
```

Excerpt of /var/log/messages on node sles10-node1:

```
Jul 30 18:44:14 sles10-node1 cibadmin: [4200]: info: Invoked: cibadmin -D
-o resources -X <primitive id="cIDS" />

Jul 30 18:44:15 sles10-node1 crmd: [3103]: info: do_lrm_rsc_op: Performing
op=cIDS_stop_0 key=32:60:ac2f114d-63f5-42a5-b132-e2c6a0fb5cd3)

Jul 30 18:44:15 sles10-node1 crmd: [3103]: info: process_lrm_event: LRM
operation cIDS_monitor_10000 (call=19, rc=-2) Cancelled

Jul 30 18:44:16 sles10-node1 cib: [4202]: info: write_cib_contents: Wrote
version 0.89.1 of the CIB to disk (digest:
298d446c266ff4b1b2e8ec014fa45e12)

Jul 30 18:44:22 sles10-node1 crmd: [3103]: info: process_lrm_event: LRM
operation cIDS_stop_0 (call=20, rc=0) complete
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected.

Results on RHEL5:

✓ As expected.

Test Case 08 (TC08)

Test Case ID: TC08

Description:

The ITVS is run with method ‘test-before’ on the node that is currently holding the IDS resource in the HA cluster. After the script rebooted the ‘failed’ node and Heartbeat failed over the resources onto another node, the ITVS is run on that node with parameter ‘test-after’.

Expected Results:

On the first node, the ITVS initiates four transactions in total and does a checkpoint of the IDS database server. After rebooting the first node and thereby forcing a

failover in the HA cluster and starting the IDS resource on a different node, only two of the four transactions should be committed. The script should therefore indicate that the validation of all four transactions is successful.

Output on SLES10:

On node sles10-node1:

```
sles10-node1:~ # cd ids-transaction-validation-script_ITVS/
sles10-node1:~/ids-transaction-validation-script_ITVS # ls
itvs.sh itvs-transaction2.sh  itvs-transaction4.sh
itvs-transaction1.sh  itvs-transaction3.sh
sles10-node1:~/ids-transaction-validation-script_ITVS # sh itvs.sh test-
before
Processing function istv_test_before
Database created.
Database closed.
Creating test database 'itvs': [ success ]
Executed transaction1 in background...
Sleeping for 10 seconds...
Executed transaction2 in background...
Sleeping for 2 seconds...
Performing IDS checkpoint...
Performing IDS checkpoint: [ success ]
Executed transaction3 in background...
Sleeping for 2 seconds...
Executed transaction4 in background...
Rebooting this cluster node in 20 seconds
(to ensure transaction2 and transaction4 were committed in the meanwhile)
in order to force resource failover of IDS.
Please run 'itvs.sh test-after' after failover on the cluster node the IDS
resource failed over to.
```

Excerpt of /Informix/logs/online.log on node sles10-node1:

```
11:31:47  Dynamically allocated new virtual shared memory segment (size
8192KB)
11:31:47  Memory sizes:resident:11904 KB, virtual:24576 KB, no SHMTOTAL
limit
11:32:06  Checkpoint Completed:  duration was 0 seconds.
11:32:06  Wed Aug  1 - loguniq 10, logpos 0x55c018, timestamp: 0xbcc15
Interval: 269

11:32:06  Maximum server connections 3
11:32:06  Checkpoint Statistics - Avg. Txn Block Time 0.000, # Txns
blocked 0, Plog used 25, Llog used 957
```

On node sles10-node2:

```
sles10-node2:~ # cd ids-transaction-validation-script_ITVS/
sles10-node2:~/ids-transaction-validation-script_ITVS # sh itvs.sh test-
after
Processing function itvs_test_after
First Transaction was not committed as expected: [ success ]
Second Transaction was committed as expected: [ success ]
Third Transaction was not committed as expected: [ success ]
Fourth Transaction was committed as expected: [ success ]
SUCCESS: All tests were successful, the resource agent behaves just as
expected! :)
Database dropped.
Successfully dropped the test database 'itvs'.
```

```
sles10-node2:~/ids-transaction-validation-script_ITVS #
```

Excerpt of /Informix/logs/online.log on node sles10-node2:

```
11:34:06  IBM Informix Dynamic Server Started.

Wed Aug  1 11:34:08 2007

11:34:08  Warning: ONCONFIG dump directory (DUMPDIR) '/tmp' has insecure
permissions
11:34:08  Event alarms enabled.  ALARMPROG =
'/informix/etc/alarmprogram.sh'
11:34:08  Booting Language <c> from module <>
11:34:08  Loading Module <CNULL>
11:34:08  Booting Language <builtin> from module <>
11:34:08  Loading Module <BUILTINNULL>
11:34:13  Dynamically allocated new virtual shared memory segment (size
8192KB)
11:34:13  Memory sizes:resident:11904 KB, virtual:16384 KB, no SHMTOTAL
limit
11:34:13  DR: DRAUTO is 0 (Off)
11:34:13  DR: ENCRYPT_HDR is 0 (HDR encryption Disabled)
11:34:13  Event notification facility epoll enabled.
11:34:13  IBM Informix Dynamic Server Version 11.10.UB7     Software
Serial Number AAA#B000000
11:34:14  Performance Advisory: The current size of the physical log
buffer is smaller than recommended.
11:34:14   Results: Transaction performance might not be optimal.
11:34:14   Action: For better performance, increase the physical log
buffer size to 128.
11:34:14  The current size of the logical log buffer is smaller than
recommended.
11:34:14  IBM Informix Dynamic Server Initialized -- Shared Memory
Initialized.

11:34:14  Physical Recovery Started at Page (1:5377).
11:34:14  Physical Recovery Complete: 30 Pages Examined, 30 Pages
Restored.
11:34:14  Logical Recovery Started.
11:34:14  10 recovery worker threads will be started.
11:34:17  Logical Recovery has reached the transaction cleanup phase.
11:34:17  Logical Recovery Complete.
          2 Committed, 0 Rolled Back, 0 Open, 0 Bad Locks

11:34:20  Dataskip is now OFF for all dbspaces
11:34:20  On-Line Mode
11:34:23  SCHAPI: Started 2 dbWorker threads.
```

Output on RHEL5:

Same as on SLES10.

Results on SLES10:

✓ As expected, all tests of the ITVS were successful.

Results on RHEL5:

✓ As expected, all tests of the ITVS were successful.

B. GNU General Public License, Version 2

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software
Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes

make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

C. Bibliography

[BBC01]	BBC article from Jan 1st 2000 on the Y2K issue at http://news.bbc.co.uk/1/hi/sci/tech/585013.stm, accessed on August 4th 2007
[CentOSkernels01]	Kernels with kernel timer set to 100 Hz provided by members of the CentOS project (http://centos.org/) at http://vmware.xaox.net/centos/5/i386/, accessed on August 15th 2007
[ChC01]	Cluster Term Definition in the Chemistry Dictionary of ChemiCool at http://www.chemicool.com/definition/cluster.html, accessed on July 16th 2007
[CygX01]	Cygwin/X, an X server port to Microsoft Windows at http://x.cygwin.com/, accessed on August 14th 2007
[DRBD01]	Project website of DRBD at http://www.drbd.org, accessed on May 28th 2007
[DRBD02]	DRBD tutorial on the DRBD website at http://www.drbd.org/documentation.html, accessed on August 13th 2007
[FAQS.org01]	Chapter on File security of the Linux Introduction of faqs.org at http://www.faqs.org/docs/linux_intro/sect_03_04.html, accessed on July 4th 2007
[FatH01]	Folding@Home project web site at http://folding.stanford.edu/, accessed on August 3rd 2007
[FroKro01]	Article about Open Clusters by Hartmut Frommert and Christine Kronberg at http://www.seds.org/messier/open.html, accessed on July 16th 2007
[Gite01]	Linux Shell Scripting Tutorial v1.05r3 by Vivek G. Gite at http://www.freeos.com/guides/lsst/, accessed on July 12th 2007
[Gos01]	Post on the Linux-HA mailing list indicating that the DRBD resource agent in Heartbeat 2.0.8 is buggy at http://www.gossamer-threads.com/lists/linuxha/users/41228#41228, accessed on August 13th 2007
[GPL01]	Version 2 of the GPL at http://www.gnu.org/licenses/gpl.html, accessed on May 28th 2007

[HP01]	HP Serviceguard for Linux product website at http://www.hp.com/solutions/enterprise/highavailability/linux/serviceguard/index.html, accessed on June 29th 2007
[IBM01]	IDS system requirements at http://www.ibm.com/software/data/informix/ids/requirements.html, accessed on June 20th 2007
[IBM02]	Informix tools at http://www.ibm.com/software/data/informix/tools/, accessed on June 20th 2007
[IBM03]	IDS features and benefits at http://www.ibm.com/software/data/informix/ids/feature.html, accessed on June 21st 2007
[IBM04]	IBM DB2 Data Server at http://www.ibm.com/software/data/db2/, accessed on June 21st 2007
[IBM05]	IBM Redbook: IBM Informix Dynamic Server V10.0 Information Center at http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp, accessed on June 22nd 2007
[IBM06]	IBM Redbook: Informix Dynamic Server V10…Extended Functionality for Modern Busyness by Chuck Ballard and his Co-Authors, December 2006, IBM Corporation, ISBN 0738494739
[IBM07]	IBM press release for introducing IDS 11 at http://www.ibm.com/press/us/en/pressrelease/21697.wss, accessed on June 22nd 2007
[IBM08]	IBM Redbook: Informix Dynamic Server V10: Superior Data Replication for Availability and Distribution by Chuck Ballard and his Co-Authors, April 2007, IBM Corporation, ISBN 0738486221
[IBM09]	IBM Informix Dynamic Server Enterprise Replication Guide, no author, December 2004, IBM Corporation, no ISBN
[IBM10]	IBM Informix Dynamic Server and High Availability on Sun Cluster 3.x by Srinivasrao Madiraju, Pradeep Kulkarni and Michael Nemesh, December 2004, IBM Corporation, no ISBN
[IBM11]	IBM HACMP product website at http://www.ibm.com/systems/p/ha/, accessed on June 29th 2007
[IBM12]	IBM developerWorks website at http://www.ibm.com/developerworks/, accessed on August 16th 2007

[IIUG01]	Website of the International Informix User Group (IIUG) at http://iiug.org/, accessed on August 16th 2007
[intek01]	Website of the German IT company innotek at http://www.innotek.de/index.php?lang=en, accessed on August 14th 2007
[IXZ01]	IBM's Data Server Positioning at http://www.informix-zone.com/ids-positioning, accessed on June 21st 2007
[IXZ02]	IDS Survey by oninit and openpsl at http://www.informix-zone.com/oninit-openpsl-survey, accessed on June 21st 2007
[Ixzone01]	Portal devoted to IDS called The Informix Zone at http://www.informix-zone.com/, accessed on August 16th 2007
[Kop01]	The Linux Enterprise Cluster by Karl Kopper, chapters 6 – 9, 2005, No Starch Press Inc., ISBN 1593270364
[Len01]	Historical timeline of the company Lenovo at http://www.pc.ibm.com/us/lenovo/about/history.html, accessed on August 14th 2007
[LHA01]	Linux-HA project website at http://www.linux-ha.org, accessed on June 29th 2007
[LHA02]	Linux-HA project wiki at http://wiki.linux-ha.org, accessed on June 29th 2007
[LHA03]	Subsection dedicated to DRBD on the Linux-HA project website at http://www.linux-ha.org/DRBD, accessed on May 28th 2007
[LHA04]	Page on the new features of Heartbeat Version 2 at http://linux-ha.org/NewHeartbeatDesign, accessed on July 2nd 2007
[LHA05]	Linux-HA Release 2 Fact Sheet at http://linux-ha.org/FactSheetv2, accessed on July 4th 2007
[LHA06]	Article discussing the haresources2cib.py script on the Linux-HA website at http://linux-ha.org/ClusterInformationBase/Conversion, accessed on July 4th 2007
[LHA07]	Article discussing simple CIB examples on the Linux-HA website at http://linux-ha.org/ClusterInformationBase/SimpleExamples, accessed on July 4th 2007
[LHA08]	XML DTD of the CIB on the Linux-HA website at http://linux-ha.org/ClusterResourceManager/DTD1.0/Annotated, accessed on July 9th 2007

[LHA09]	Article explaining the clone resource type on the Linux-HA website at http://www.linux-ha.org/v2/Concepts/Clones, accessed on July 9th 2007
[LHA10]	Article explaining the master/slave resource type on the Linux-HA website at http://www.linux-ha.org/v2/Concepts/MultiState, accessed on July 9th 2007
[LHA11]	Article on Dunn's Law of Information on the Linux-HA project website at http://linux-ha.org/DunnsLaw, accessed in August 9th 2007
[LHA12]	Article on STONITH on the Linux-HA website at http://linux-ha.org/STONITH, accessed on August 9th 2007
[LHA13]	Article on split-brain on the Linux-HA website at http://linux-ha.org/SplitBrain, accessed on August 9th 2007
[LHA14]	Article on fencing on the Linux-HA website at http://linux-ha.org/Fencing, accessed on August 9th 2007
[LHA15]	Article on major cluster concepts on the Linux-HA website at http://www.linux-ha.org/ClusterConcepts, accessed on August 10th 2007
[LHA16]	Article on ping nodes on the Linux-HA website at http://www.linux-ha.org/PingNode, accessed on August 10th 2007
[LHA17]	Article explaining the basic architecture of Heartbeat Version 2 on the Linux-HA website at http://www.linux-ha.org/BasicArchitecture, accessed on July 11th 2007
[LHA18]	Article briefly linking the new design aspects of Heartbeat version 2 on the Linux-HA website at http://www.linux-ha.org/NewHeartbeatDesign, accessed on July 12th 2007
[LHA19]	Article on resource agents supported by Heartbeat version 2 on the Linux-HA website at http://www.linux-ha.org/ResourceAgent, accessed on July 12th 2007
[LHA20]	Article on Heartbeat resource agents on the Linux-HA website at http://www.linux-ha.org/HeartbeatResourceAgent, accessed on July 12th 2007
[LHA21]	Article on LSB resource agents on the Linux-HA website at http://www.linux-ha.org/LSBResourceAgent, accessed on July 12th 2007
[LHA22]	Article on OCF resource agents on the Linux-HA website at http://www.linux-ha.org/OCFResourceAgent, accessed on July 12th 2007

[LHA23]	Article on how to use DRBD in Heartbeat version 2 configuration mode at http://linux-ha.org/DRBD/HowTov2, accessed on August 13th 2007
[LHA24]	Article on possible resources for Linux-HA support at http://linux-ha.org/ContactUs, accessed on August 13th 2007
[LHA25]	Article on quorum servers on the Linux-HA website at http://www.linux-ha.org/QuorumServerGuide, accessed on August 15th 2007
[LHA26]	Article on split-site with a section on quorum plugins on the Linux-HA website at http://www.linux-ha.org/SplitSite, accessed on August 15th 2007
[LHAbug#1630]	Bug report for the Linux-HA bug #1630 at http://old.linux-foundation.org/developer_bugzilla/show_bug.cgi?id=1630, accessed on August 15th 2007
[LHAbug#1661]	Bug report for Linux-HA bug #1661 at http://old.linux-foundation.org/developer_bugzilla/show_bug.cgi?id=1661, accessed on August 15th 2007
[LHAdev01]	The official Linux-HA development repository at http://hg.linux-ha.org/dev/, accessed on August 16th 2007
[LHAdevlist01]	Discussion on the Linux-HA dev mailing list relevant to the Linux-HA bug #1661 at http://lists.community.tummy.com/pipermail/linux-ha-dev/2007-July/014765.html, accessed on August 15th 2007
[LHAmlist01]	Thread introducing the interim Heartbeat packages provided by Andrew Beekhof on the general Linux-HA mailing list at http://www.gossamer-threads.com/lists/linuxha/users/41581?search_string=ids, accessed on August 16th 2007
[LHAnet01]	LinuxHA.net project website at http://www.linuxha.net, accessed on June 29th 2007
[LINBIT01]	Company website of the DRBD authors at http://www.linbit.com, accessed on May 28th 2007
[LSB01]	Project website of the Linux Standard Base at http://www.linux-foundation.org/en/LSB, accessed on July 12th 2007
[LVS01]	Linux Virtual Server project web site at http://www.linuxvirtualserver.org/, accessed on August 3rd 2007
[MaSte01]	Blueprints for High-Availability by Evan Marcus and Hal Stern, Second Edition, 2003, Wiley Publishing Inc., ISBN 0471430269

[OCAL01]	Rabbit clipart used in one of the figures taken from the project website of the Open Clip Art Library at http://openclipart.org/people/danko/danko_Friendly_rabbit.svg, accessed on August 7th 2007
[OCF01]	Draft of the OCF specifications at http://www.opencf.org/cgi-bin/viewcvs.cgi/specs/ra/resource-agent-api.txt?rev=HEAD, accessed on July 12th 2007
[Ossh01]	Manpage for the configuration file of the Open SSH server at http://www.openbsd.org/cgi-bin/man.cgi?query=sshd_config, accessed on August 14th 2007
[OSSI01]	OpenSSI project website at http://www.openssi.org, accessed on June 28th 2007
[OSSI02]	OpenSSI project wiki at http://wiki.openssi.org, accessed on June 28th 2007
[OSSI03]	Web-based monitoring GUI for OpenSSI at http://openssi-webview.sourceforge.net, accessed on June 28th 2007
[Pfi01]	In Search of Clusters by Gregory F. Pfister, Second Edition, 1998, Prentice-Hall Inc., ISBN 0138997098
[Pla01]	"DRBD in a Heartbeat" by Pedro Pla, published June 26th 2006 at http://www.linuxjournal.com/article/9074, accessed at May 28th 2007
[Q1]	Quotation from http://www.quotegarden.com/computers.html, Author unknown, accessed on August 10th 2007
[Q2]	Murphy's Computer Laws from http://www.murphys-laws.com/murphy/murphy-computer.html, Author unknown, accessed on August 10th 2007
[Q3]	Wyland's Law of Automation from http://en.wikiquote.org/wiki/Automation, Author unknown, accessed on August 10th 2007
[Q4]	Quotation from http://en.wikiquote.org/wiki/Henry_Ford, Henry Ford, accessed on August 10th 2007
[RedHatbug#212180]	Bug report for the Red Hat bug #212180 at https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=212180, accessed on August 15th 2007
[RFC1321]	MD5 defined in RFC1321 at http://tools.ietf.org/html/rfc1321, accessed on July 4th 2007

[RFC3174]	SHA1 defined in RFC 3174 at http://tools.ietf.org/html/rfc3174, accessed on July 4th 2007
[RH01]	Red Hat Cluster Suite product website at http://www.redhat.com/cluster_suite/, accessed on June 29th 2007
[RoBe01]	Classic Shell Scripting by Arnold Robbins and Nelson H. F. Beebe, May 2005, O'Reilly Media, ISBN 0596005954
[Roy01]	Whitepaper "Comparing IDS 10.0 and Oracle 10g" by Jackes Roy, October 2005, no ISBN
[Ruby01]	Ruby/Informix project website at http://ruby-informix.rubyforge.org/index.html, accessed on June 20th 2007
[S11N01]	News archive on the terrorist attacks on September 11th 2001 at http://www.september11news.com/, accessed on August 4th 2007
[SNIA01]	Dictionary of the Storage Networking Industry Association (SNIA) at http://www.snia.org/education/dictionary/, accessed on August 5th 2007
[Sun01]	Datasheet Sun Plex Systems and Sun Cluster 3.0 Software, no author, 2002, Sun Microsystems, no ISBN
[Sun02]	Documentation Center for Sun Cluster 3.2 at http://docs.sun.com/app/docs/doc/820-0335/6nc35dge2?a=view, accessed on May 28th 2007
[Ubuntu01]	Website of Ubuntu Linux at http://www.ubuntu.com/, accessed on August 14th 2007
[UM01]	Project website of Ultra Monkey at http://www.ultramonkey.org, accessed on June 28th 2007
[VBox01]	Project website of the virtualization software VirtualBox at http://www.virtualbox.org/, accessed on August 14th 2007
[VBox02]	Article on virtualization in general on the VirtualBox website at http://www.virtualbox.org/wiki/Virtualization, accessed on August 14th 2007
[VBox03]	Article on VirtualBox on the VirtualBox website at http://www.virtualbox.org/wiki/VirtualBox, accessed on August 14th 2007
[VBox04]	VirtualBox user manual available at http://virtualbox.org/download/UserManual.pdf, accessed on August 14th 2007

[VBox05]	Documentation section on the VirtualBox website at http://virtualbox.org/wiki/Documentation, accessed on August 14th 2007
[VBoxbug#521]	Bug report for the VirtualBox bug #521 at http://www.virtualbox.org/ticket/521, accessed on August 15th 2007
[VBoxforums01]	Thread discussing the issue of VMs that produce too much system load on the host in the VirtualBox forums at http://forums.virtualbox.org/viewtopic.php?t=507, accessed on August 15th 2007
[Ver01]	Veritas Cluster Server product website at http://www.symantec.com/enterprise/products/overview.jsp?pcid=1019&pvid=20_1, accessed on June 29th 2007
[WCG01]	World Community Grid project web site at http://www.worldcommunitygrid.org/, accessed on August 8th 2007
[Wikip01]	Article about CRC on Wikipedia.org at http://en.wikipedia.org/wiki/Cyclic_redundancy_check, accessed on July 4th 2007

D. CD-ROM

In the upper most directory of the CD-ROM, the file *README.txt*, containing the following text, is found:

This is the CD-ROM that accompanies the diploma thesis of Lars D. Forseth (lars@forseth.de) and has the theme “High-Availability Cluster Support for IBM Informix Dynamic Server (IDS) on Linux”. This file explains the folder structure on the CD-ROM.

An explanation what to find in each folder follows:

- **brainstormings**,
  contains the brainstormings that were made as a preparation for each chapter of the thesis.
- **chapters**,
  contains the original files for each chapter of the thesis, including the original figures before they were imported into Microsoft Word.
- **covers**,
  contains the original files for the cover of the thesis itself and its four parts (I - IV).
- **development-cluster-configs**,
  contains the configuration files used for the test cluster in the development environment.
- **diploma-thesis-FINAL**,
  contains the above mentioned diploma thesis as a whole as .doc and .pdf files.
- **IDS-RA-FINAL**,
  contains the final version of the source code of the IDS resource agent for Heartbeat.

- **ids-transaction-validation-script_ITVS**,
  contains the source code of the IDS Transaction Validation Script (ITVS) used for the eighth test case (TC08) during the validation process.

- **Installation_Guide_for_SLES10_and_RHEL5**,
  contains the installation guide describing how to setup the cluster on SLES10 and RHEL5 used in the validation process. The original figures, before they were imported into Microsoft Word, are also included.

- **license**,
  contains a copy of the license under which the IDS RA is published: GNU General Public License, Version 2.

- **specs**,
  contains the specification files (non-functional requirements, functional requirements, design specification and test cases) that were created during the project for this thesis. The milestones plan of the project is also included.

- **validation-cluster-configs**,
  contains the configuration files used for the test cluster in the validation environment. These files are useful when following the installation guide also contained on this CD-ROM.

- **vbox-scripts**,
  contains shell scripts that encapsulate the most common commands needed to manage the VirtualBox VMs during the validation process.